SAS® Viya™ 3.1 Data Management and Utility Procedures Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® Viya™ 3.1 Data Management and Utility Procedures Guide*. Cary, NC: SAS Institute Inc.

SAS® Viya™ 3.1 Data Management and Utility Procedures Guide

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

September 2016

3.1-P1:proc

Contents

Part 1

Procedures

Chapter 1

APPEND Procedure

Overview: APPEND Procedure

Using the APPEND procedure with the CAS engine, you can do the following:

- add rows from a CAS table to the end of a SAS data set.
- add rows from a SAS data set to the end of another SAS data set.

Using the APPEND procedure, you cannot do the following:

- add rows from a CAS table to the end of another CAS table.
- add rows from a SAS data set to the end of a CAS table.

The BASE= option or OUT= option cannot specify a CAS table.

Generally, the APPEND procedure functions the same as the APPEND statement in the DATASETS procedure. The only difference between the APPEND procedure and the APPEND statement in PROC DATASETS is the default for *libref* in the BASE= and DATA= options. The default for the APPEND procedure for *libref* in the BASE= and DATA= options is either Work or User. For the APPEND statement, the default is the libref of the procedure input library.

Concepts: APPEND Procedure

Using the Block I/O Method to Append

Note: The block I/O method cannot be used when appending a CAS table to a SAS data set.

The block I/O method is used to append blocks of data instead of one row at a time. This method increases performance when you are appending large data sets. The software determines whether to use the block I/O method. Not all data sets can use the block I/O method. There are restrictions set by the APPEND procedure and the BASE engine.

To display information in the log about the append method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

The following message is written to the log, if the block I/O method is not used:

```
INFO: Data set block I/O cannot be used because:
```

If the APPEND procedure determines that the block I/O will not be used, one of the following explanations is written to the log:

```
INFO: - The data sets use different engines, have different
variables or have attributes that might differ.
```

```
INFO: - There is a WHERE clause present.
```

```
INFO: - There is no member level locking.
```

If the BASE engine determines that the block I/O method will not be used, one of the following explanations is written to the log:

```
INFO: - Cross Environment Data Access is being used.
```

```
INFO: - The file is compressed.
```

Restricting the Rows That Are Appended

You can use the WHERE statement in order to restrict the rows from the DATA= CAS table. The WHERE statement has no effect on the BASE= SAS data set. Similarly, you can use the WHERE= data set option with the DATA= table in order to restrict the rows that are appended, except when appending a data set to itself. If you use the WHERE= data set option with the BASE= SAS data set, it affects which rows become appended.

- **For an existing BASE= SAS data set:** The WHERE= data set option takes effect only if the WHEREUP= data set option is set to YES.
- **For the non-existent BASE= SAS data set:** The WHERE= data set option takes effect regardless of the WHEREUP= data set option setting.

Choosing between the SET Statement and the APPEND Statement

If you use the SET statement in a DATA step to concatenate two SAS data sets, SAS must process all the rows in both data sets to create a new one. The APPEND statement

bypasses the processing of data in the original data set and adds new rows directly to the end of the original data set. Using the APPEND procedure can be more efficient than using a SET statement if the BASE= SAS data set is large.

The APPEND procedure is especially useful if you frequently add rows to a SAS data set (for example, in production programs that are constantly appending data to a journal-type data set).

Appending to a Compressed Data Set

You can concatenate compressed SAS data sets and CAS tables. Either or both of the BASE= SAS data set and DATA= data set or CAS table can be compressed. If the BASE= data set allows the reuse of space from deleted rows, the APPEND statement might insert the rows into the middle of the BASE= data set.

For information about the COMPRESS= and REUSE= data set and system options, see *SAS Viya Data Set Options: Reference* and *SAS Viya System Options: Reference*.

Appending to Data Sets with Different Variables

If the DATA= SAS data set or CAS table contains variables that are not in the BASE= SAS data set, use the FORCE option in the APPEND procedure. This option forces the concatenation of the two SAS data sets or a CAS table to a SAS data set. The APPEND procedure drops the extra variables and issues a warning message. You can use the NOWARN option to suppress the warning message.

If the BASE= SAS data set contains a variable that is not in the DATA= SAS data set or CAS table, the APPEND procedure concatenates. But, the rows from the DATA= data set or table have a missing value for the variable that was not present in the DATA= data set or table. The FORCE option is not necessary in this case.

If you use the DROP=, KEEP=, or RENAME= options on the BASE= SAS data set, the options ONLY affect the APPEND processing. It does not change the variables in the appended BASE= SAS data set. Variables that are dropped or not kept using the DROP= and KEEP= options still exist in the appended BASE= SAS data set. Variables that are renamed using the RENAME= option remain with their original name in the appended BASE= SAS data set.

Appending to Data Sets That Contain Variables with Different Attributes

If a variable has different attributes in the BASE= SAS data set than it does in the DATA= data set or table, the attributes in the BASE= data set prevail.

If the formats in the DATA= SAS data set or CAS table are different from those in the BASE= SAS data set, then the formats in the BASE= data set are used. However, SAS does not convert the data from the DATA= data set or table in order to be consistent with the SAS formats in the BASE= data set. The result could be data that seems to be incorrect. A warning message is displayed in the log.

Use the FORCE option if one of the following occurs:

- if the length of a variable is longer in the DATA= SAS data set or CAS table than in the BASE= SAS data set
- if the same variable is a character variable in one data set or table and a numeric variable in the other

Using FORCE has the following consequences:

- The length of the variables in the BASE= SAS data set takes precedence. The values might be truncate from the DATA= data set or CAS table to fit them into the length that is specified in the BASE= data set.
- The type of the variables in the BASE= data set takes precedence. The APPEND procedure replaces values of the wrong type (all values for the variable in the DATA= data set or table) with missing values.

System Failures

If a system failure or some other type of interruption occurs while the procedure is executing, the Append operation might not be successful; it is possible that not all, perhaps none, of the rows are added to the BASE= SAS data set. In addition, the BASE= data set might suffer damage. The Append operation performs an update in place, which means that it does not make a copy of the original data set before it begins to append rows.

Syntax: APPEND Procedure

Restrictions: BASE= option and OUT= option cannot be a CAS table.

You cannot concatenate two CAS tables.

Requirement: The BASE= SAS data set must be a member of a library that supports update processing.

Tips: You can specify some data set options for the BASE= argument and DATA= option. However, if you specify DROP=, KEEP=, or RENAME= data set option for the BASE= SAS data set, the option is ignored. You can use any global statements as well.

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements.

PROC APPEND BASE=*<libref.>SAS-data-set*
<DATA=*<libref.>data-set*>
<FORCE>
<NOWARN>;

Statement	Task	Example
PROC APPEND	Add rows from a SAS data set or a CAS table to the end of another SAS data set	Ex. 1

PROC APPEND Statement

Add rows from a SAS data set or CAS table to the end of another SAS data set.

Syntax

PROC APPEND BASE=*<libref.>data-set*

<DATA=<*libref.*>*data-set*>
<option(s)>;

Required Argument

BASE=<*libref.*> *data-set*
names the SAS data set to which you want to add rows. The BASE= option cannot be a CAS table.

libref
specifies the library that contains the SAS data set. If you omit the libref, the default is the libref for the procedure input library. If you are using PROC APPEND, the default for libref is either Work or User.

data-set
names a SAS data set. If the APPEND procedure cannot find an existing data set with this name, it creates a new data set in the library. That is, you can use the APPEND procedure to create a SAS data set by specifying a new SAS data set name in the BASE= argument.

Whether you are creating a new data set or appending to an existing data set, the BASE= SAS data set is the current data set after all Append operations.

Alias OUT=

Optional Arguments

DATA=<*libref.*> *data-set*
names the SAS data set or CAS table containing rows that you want to append to the end of the data set specified in the BASE= argument.

libref
specifies the library that contains the data set or table. If you omit libref, the default is the libref for the procedure input library. The DATA= data set or table can be from any library. You must use the two-level name if the data set resides in a library other than the procedure input library.

data-set
names a data set or table. If the APPEND procedure cannot find an existing data set with this name, it stops processing.

Alias NEW=

Default the most recently created data set, from any library

FORCE
forces the APPEND statement to concatenate SAS data sets or a CAS table to a SAS data set when the DATA= data set or table contains variables that meet one of the following criteria:

- are not in the BASE= SAS data set
- do not have the same type as the variables in the BASE= SAS data set
- are longer than the variables in the BASE= SAS data set

NOWARN
suppresses the warning when used with the FORCE option to concatenate two SAS data sets or a CAS table and a SAS data set with different variables.

Using the APPEND Procedure

To copy only the table metadata and structure of a data set but not the data, use the following example where Dataset1 is nonexistent:

```
proc append base=dataset1 data=dataset2(obs=0);
run;

proc contents data=dataset1;
run;
quit;
```

Example: Concatenating a CAS Table to a SAS Data Set

Features: PROC APPEND statement options
BASE=
DATA=
FORCE

Other features: OPTIONS statement
CONTENTS procedure

Details

This example demonstrates the following tasks:

- appending a CAS table to a SAS data set
- contents of the table, the data set, and the new data set after appending

Program

```
options pagesize=40 linesize=64 nodate pageno=1;

libname sascas1 cas;

libname saleslib 'directory-name';

proc contents data=saleslib.monthly;
run;

proc contents data=sascas1.lastmonth;
run;

proc append base=saleslib.monthly data=sascas1.lastmonth force;
run;

proc sql outobs=5;
   select store_id, address, city, state, zipcode, totalsales
```

```
   format dollar12.
   from saleslib.monthly(obs=4)
   where totalsales gt 2000000;
quit;
```

Program Description

This example appends a CAS table to the end of a SAS data set.

Set the system options. The NODATE option suppresses the display of the date and time in the output. The PAGENO= option specifies the starting page number. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page.

```
options pagesize=40 linesize=64 nodate pageno=1;
```

The LIBNAME statements assign the CAS engine and BASE engine libraries.

```
libname sascas1 cas;

libname saleslib 'directory-name';
```

Check the contents of the table and data set. Use PROC CONTENTS to view the data set and table.

```
proc contents data=saleslib.monthly;
run;

proc contents data=sascas1.lastmonth;
run;
```

Append the SasCas1.LastMonth table to the SalesLib.Monthly data set. The data for last month's sales in a CAS table is appended to the accumulated sales data stored in a SAS data set. The CAS table uses VARCHAR to store the city and address values. The SAS data set stores the values in character variables. Since the attribute for the two values differ, the FORCE option is used in PROC APPEND.

```
proc append base=saleslib.monthly data=sascas1.lastmonth force;
run;
```

Retrieve total sales. Use PROC SQL to retrieve five variables and sales that are greater than $2,000,000.

```
proc sql outobs=5;
   select store_id, address, city, state, zipcode, totalsales
   format dollar12.
   from saleslib.monthly(obs=4)
   where totalsales gt 2000000;
quit;
```

Warnings in the Log

Note the warnings that were sent to the log.

```
117        proc append base=saleslib.monthly data=sascas1.lastmonth force;
 118        run;

 NOTE: Appending SASCAS1.LASTMONTH to SALESLIB.MONTHLY.
 WARNING: Variable city has different lengths on BASE and DATA files
         (BASE 100 DATA 21).
 WARNING: Variable address has different lengths on BASE and DATA files
         (BASE 160 DATA 19).
 NOTE: There were 12 observations read from the data set SASCAS1.LASTMONTH.
 NOTE: 12 observations added.
 NOTE: The data set SALESLIB.MONTHLY has 24 observations and 7 variables.
```

Output: Concatenating a CAS Table to a SAS Data Set

Output 1.1 *The CAS Table Contents*

LastMonth Table

The CONTENTS Procedure

Data Set Name	SASCAS1.LASTMONTH	**Observations**	12
Member Type	DATA	**Variables**	7
Engine	CAS	**Indexes**	0
Created	08/11/2016 10:27:58	**Observation Length**	80
Last Modified	08/11/2016 10:27:58	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
6	address	Varchar	40
4	city	Varchar	25
7	month	Char	10
5	state	Char	2
1	store_id	Char	3
2	totalsales	Num	8
3	zipcode	Char	5

Output 1.2 *The Monthly Data Set Contents*

Monthly Data Set

The CONTENTS Procedure

Data Set Name	SALESLIB.MONTHLY	**Observations**	12
Member Type	DATA	**Variables**	7
Engine	V9	**Indexes**	0
Created	08/11/2016 09:27:58	**Observation Length**	288
Last Modified	08/11/2016 09:27:58	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	227
Obs in First Data Page	12
Number of Data Set Repairs	0
Filename	/r/ge.unx.sas.
Release Created	V.0301M0
Host Created	Linux
Inode Number	73976242
Access Permission	rw-r--r--
Owner Name	
File Size	128KB
File Size (bytes)	131072

Output 1.3 *Concatenated Data Set*

Concatenated Data Set

The CONTENTS Procedure

Data Set Name	SALESLIB.MONTHLY	**Observations**	24
Member Type	DATA	**Variables**	7
Engine	V9	**Indexes**	0
Created	08/11/2016 09:27:58	**Observation Length**	288
Last Modified	08/11/2016 09:27:58	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Output 1.4 *Total Sales*

store_id	address	city	state	zipcode	totalsales
843	318 S Barnes St	What Cheer	IA	50268	$2,218,497
486	2345 Lindeman Ln	Nameless	TX	78641	$2,101,480
814	115 Tuscarora Pike	Shanghai	WV	25427	$2,195,378
842	318 S Lamar St	What Cheer	IA	50268	$2,218,497

Chapter 2

CONTENTS Procedure

Overview: CONTENTS Procedure

The CONTENTS procedure shows the contents of a CAS table and prints the directory of the caslib. The following code prints Mycas.Cars:

```
proc contents data=mycas.cars;
run;
```

Output 2.1 *Mycas Caslib and Mycas.CarsTable*

The CONTENTS Procedure

Data Set Name	MYCAS.CARS	**Observations**	428
Member Type	DATA	**Variables**	15
Engine	CAS	**Indexes**	0
Created	08/11/2016 10:56:31	**Observation Length**	160
Last Modified	08/11/2016 10:56:31	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Use the following code to show the contents of the Mycas caslib:

```
proc contents data=mycs.cars directory details;
run;
```

Using the DIRECTORY and DETAILS Options

The CONTENTS Procedure

Directory	
Libref	MYCAS
Engine	CAS
Physical Name	37bada30
Server Session UUID	37bada30
Server Session Name	CASAUTO
Server Host	10.21.19.167
Server Session Port	5570
Server Session CASLIB	CASUSER

#	Name	Member Type	Obs, Entries or Indexes	Vars	Label	Number of Rows	Number of Columns	Last Modified	Data Encoding	Scope
1	AIR	DATA	144	2		144	2	08/11/2016 11:10:12	utf-8	Session
2	AIR2	DATA	144	2		144	2	08/11/2016 11:10:13	utf-8	Session
3	CARS	DATA	428	15		428	15	08/11/2016 12:55:00	utf-8	Session
4	CARSOUT	DATA	15	41		15	41	08/11/2016 12:33:18	utf-8	Session
5	CLASS	DATA	19	5	hello	19	5	08/11/2016 11:10:12	utf-8	Session
6	LASTMONTH	DATA	12	7		12	7	08/11/2016 10:27:58	utf-8	Session

Generally, the CONTENTS procedure functions the same as the CONTENTS statement in the DATASETS procedure. The default for *libref* in the DATA= option in the CONTENTS procedure is Work. For the CONTENTS statement, the default is the libref of the procedure input library.

Concepts: CONTENTS Procedure

Printing Variables

The CONTENTS statement prints an alphabetical listing of the variables by default with uppercase and lowercase names listed separately, except for variables in the form of a numbered range list. Numbered range lists, such as x1–x100, are printed in incrementing order, that is, x1–x100.

Note: If a label is changed after a view is created from a table with variable labels, the CONTENTS procedure output shows the original labels.

Syntax: CONTENTS Procedure

Restriction: You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any rows.

Tips: You can use data set options with the DATA= option and the OUT= option. options.

The ORDER= option does not affect the order of the OUT= table.

PROC CONTENTS DATA=*CAS-table-specification*;

<DETAILS | NODETAILS>

<DIRECTORY>

<FMTLEN>

<NODS>
<NOPRINT>
<ORDER=COLLATE | CASECOLLATE | IGNORECASE | VARNUM>
<OUT=*table-name*>
<SHORT>
<VARNUM>;

Statement	Task	Example
PROC CONTENTS	List the contents of one or more CAS tables and print the directory of the caslib	Ex. 1, Ex. 2, Ex. 3

PROC CONTENTS Statement

List the contents of one or more CAS tables and prints the directory of the caslib.

Syntax

PROC CONTENTS DATA=*CAS-table-specification* <*option(s)*>;

Required Argument

DATA=*CAS-table-specification*
specifies an entire caslib or a specific table within a caslib. *CAS-table-specification* can take one of the following forms:

<libref.>CAS-table
names one CAS table to process. The default for libref is the libref of the procedure input library. For example, to obtain the contents of the table HtWt from the procedure input library, use the following PROC CONTENTS:

```
proc contents data=HtWt;
```

***<libref.>*_ALL_**
gives you information about all tables.*libref* refers to the caslib. The default for libref is the libref of the procedure input library. DATA=_ALL_ automatically prints a listing of the tables that are contained in the caslib.

Default most recently created table in your job or session, from any library.

Optional Arguments

CLONE | NOCLONE
specifies whether to copy data set attributes. The only attribute that can be used with the CAS engine is COMPRESS.

Attributes are specified with data set options, system options, or LIBNAME statement options. The CAS engine supports only the COMPRESS=YES | NO option. No other attributes are supported by the CAS engine.

The following table summarizes how the COPY statement works:

Table 2.1 *CLONE Interaction with Attributes*

Attribute	To	CLONE or NOCLONE	Description
BUFSIZE= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	OVERRIDE=(BUFSIZE=*value other than default*)
		NOCLONE	Uses setting of BUFSIZE= system option
	CAS table to CAS table		
COMPRESS= SAS data set - COMPRESS=BINARY | CHAR | NO CAS table - COMPRESS=NO | YES	SAS data set to CAS table	CLONE	A compressed SAS data set becomes a compressed CAS table unless OVERRIDE= is used. A not compressed SAS data set becomes a not compressed CAS table unless OVERRIDE= is used.
		NOCLONE	Follows the CAS LIBNAME setting.
	CAS table to SAS data set	CLONE	Compressed CAS table becomes a SAS data set CHAR unless the OVERRIDE= is used.
		NOCLONE	The COMPRESS= system option or LIBNAME option value is used.
	CAS table to CAS table	CLONE	Keeps the current setting.
		NOCLONE	Set using the CAS LIBNAME setting for the OUT= libref.
REUSE= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	REUSE=NO unless OVERRIDE= or REUSE=YES system option is used.

Attribute	To	CLONE or NOCLONE	Description
		NOCLONE	Uses the REUSE= system option value.
	CAS table to CAS table		
POINTOBS= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	POINTOBS=NO unless OVERRIDE= is used.
		NOCLONE	POINTOBS=NO, if the CAS table is compressed and the LIBNAME statement has POINTOBS=NO. POINTOBS=YES, if the CAS table is compressed and the LIBNAME option is missing.
	CAS table to CAS table		
OUTREP= CAS engine does not support.	SAS data set to CAS table	CLONE	Converts to LINUX_86_64 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)
		NOCLONE	Converts to LINUX_86_64 if needed.
	CAS table to SAS data set		
		CLONE	Keeps data representation. (A warning is sent to the log if the OVERRIDE= option is used.)
	CAS table to CAS table		

Attribute	To	CLONE or NOCLONE	Description
ENCODING= CAS engine supports UTF-8 only. Changing encoding is not implemented for the CAS engine.	SAS data set to CAS table	CLONE	Converts to UTF-8 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)
		NOCLONE	Converts to UTF-8 if needed.
	CAS table to SAS data set	CLONE	Keeps the UTF-8 encoding unless OVERRIDE= is used.
		NOCLONE	Keeps the UTF-8 encoding unless OUTENCODING= is used in the output data set LIBNAME is used.
	CAS table to CAS table		

DETAILS | NODETAILS
includes information in the output about the number of rows, number of variables, and table labels. DETAILS includes additional columns of information in the output, but only if DIRECTORY is also specified.

Default If neither DETAILS nor NODETAILS is specified, the default for the CONTENTS procedure is the system option setting, which is NODETAILS; for the CONTENTS statement, the default is whatever is specified in the PROC DATASETS statement, which also defaults to the system option setting.

DIRECTORY
prints a list of all CAS tables in the specified library. If DETAILS is also specified, using DIRECTORY causes the additional columns.

FMTLEN
prints the length of the informat or format. If you do not specify a length for the informat or format when you associate it with a variable, the length does not appear in the output of PROC CONTENTS unless you use the FMTLEN option. The length also appears in the FORMATL or INFORML variable in the output table.

NODS
suppresses printing the contents of individual CAS tables when you specify _ALL_ in the DATA= option. The CONTENTS procedure prints only the caslib directory. You cannot use the NODS option when you specify only one table in the DATA= option.

NOPRINT
suppresses printing the output of the CONTENTS procedure.

ORDER=COLLATE | CASECOLLATE | IGNORECASE | VARNUM

COLLATE
prints a list of variables in alphabetical order beginning with uppercase and then lowercase names.

CASECOLLATE
prints a list of variables in alphabetical order even if they include mixed-case names and numerics.

IGNORECASE
prints a list of variables in alphabetical order ignoring the case of the letters.

VARNUM
is the same as the VARNUM option.

See "VARNUM" on page 20

Note The ORDER= option does not affect the order of the OUT= table.

OUT=*table-name*
names an output table.

Tip OUT= does not suppress the printed output from the statement. If you want to suppress the printed output, you must use the NOPRINT option.

SHORT
prints only the list of variable names for the table.

Restriction If the list of variables is more than 32,767 characters, the list is truncated and a WARNING is written to the log. To get a complete list of the variables, request an alphabetical listing of the variables.

VARNUM
prints a list of the variable names in the order of their logical position in the table. The physical position of the variable in the table is engine-dependent.

Results: CONTENTS Procedure

Output Tables

The CONTENTS Procedure

The CONTENTS procedure generates output tables. The CONTENTS procedure outputs file size in KB, MB, or GB, as appropriate. This value is an approximation and is sufficient for most purposes. The CONTENTS procedure also outputs the exact file size in bytes.

The OUT= Table

The OUT= option in the CONTENTS procedure creates an output table. Each variable in each DATA= table has one row in the OUT= table. Here are the variables in the output table:

CHARSET
: Blank

COLLATE
: Blank

COMPRESS
: indicates whether the table is compressed.

CRDATE
: date the table was created.

DELOBS
: number of rows marked for deletion in the table.

ENCRYPT
: indicates whether the table is encrypted.

ENGINE
: name of the method used to read from and write to the table.

FLAGS
: - - -

FORMAT
: variable format. The value of FORMAT is a blank if you do not associate a format with the variable.

FORMATD
: number of decimals that you specify when you associate the format with the variable. The value of FORMATD is 0 if you do not specify decimals in the format.

FORMATL
: format length. If you specify a length for the format when you associate the format with a variable, the length that you specify is the value of FORMATL. You do not specify a length for the format when you associate the format with a variable,. The value of FORMATL is the default length of the format if you use the FMTLEN option and 0 if you do not use the FMTLEN option.

GENMAX
: 0

GENNEXT
: .

GENNUM
: .

IDXCOUNT
: 0

IDXUSAGE
: None

INFORMAT
: variable informat. The value is a blank if you do not associate an informat with the variable.

INFORMD
: number of decimals that you specify when you associate the informat with the variable. The value is 0 if you do not specify decimals when you associate the informat with the variable.

INFORML
: informat length. If you specify a length for the informat when you associate the informat with a variable, the length that you specify is the value of INFORML. You do not specify a length for the informat when you associate the informat with a variable. The value of INFORML is the default length of the informat if you use the FMTLEN option and 0 if you do not use the FMTLEN option.

JUST
: justification (0=left, 1=right).

LABEL
: variable label (blank if none given).

LENGTH
: variable length.

LIBNAME
: libref used for the caslib.

MEMLABEL
: label for this SAS table (blank if no label).

MEMNAME
: table that contains the variable.

MEMTYPE
: library member type (DATA).

MODATE
: date the table was last modified.

NAME
: variable name.

NOBS
: number of rows in the table.

NODUPKEY
: indicates whether the NODUPKEY option was used in a PROC SORT statement to sort the input table.

NODUPREC
: indicates whether the RECS option was used in a PROC SORT statement.

NPOS
: physical position of the first character of the variable in the table.

POINTOBS
: indicates whether the table can be addressed by row.

PROTECT
: - - -

REUSE
: indicates whether the space made available when rows are deleted from a compressed table should be reused. If the table is not compressed, the REUSE variable has a value of NO.

SORTED
: the value depends on the sorting characteristics of the input table. Here are some possible values:

  . (period)
  : for not sorted.

0
for sorted but not validated.

1
for sorted and validated.

SORTEDBY
the value depends on that variable's role in the sort. Here are some possible values:

. (period)
if the variable was not used to sort the input table.

n
where *n* is an integer that denotes the position of that variable in the sort. A negative value of *n* indicates that the table is sorted by the descending order of that variable.

TRANSCODE
indicates whether the variable is transcoded.

TYPE
type of the variable (1=numeric, 2=character, 6=varchar).

TYPEMEM
special table type (blank if no TYPE= value is specified).

VARNUM
variable number in the table. Variables are numbered in the order in which they appear.

The output table is sorted by the variables LIBNAME and MEMNAME.

Note: The variable names are sorted so that the values X1, X2, and X10 are listed in that order, not in the true collating sequence of X1, X10, X2.

Here is an example of an output table created from the Mycas.Cars table.

Output 2.2 *An Example of an Output Table*

Obs	LIBNAME	MEMNAME	MEMLABEL	TYPEMEM	NAME	TYPE	LENGTH	VARNUM	LABEL	FORMAT
1	MYCAS	CARS			Cylinders	1	8	14		
2	MYCAS	CARS			DriveTrain	2	5	5		
3	MYCAS	CARS			EngineSize	1	8	8	Engine Size (L)	
4	MYCAS	CARS			Horsepower	1	8	15		
5	MYCAS	CARS			Length	1	8	9	Length (IN)	
6	MYCAS	CARS			MPG_City	1	8	10	MPG (City)	
7	MYCAS	CARS			MPG_Highway	1	8	11	MPG (Highway)	
8	MYCAS	CARS			MSRP	1	8	7		DOLLAR
9	MYCAS	CARS			Make	2	13	1		
10	MYCAS	CARS			Model	6	40	2		
11	MYCAS	CARS			Origin	2	6	4		
12	MYCAS	CARS			Type	2	8	3		
13	MYCAS	CARS			Weight	1	8	12	Weight (LBS)	
14	MYCAS	CARS			Wheelbase	1	8	13	Wheelbase (IN)	
15	MYCAS	CARS			invoice	1	8	6		DOLLAR

FORMAT	FORMATL	FORMATD	INFORMAT	INFORML	INFORMD	JUST	NPOS	NOBS	ENGINE	CRDATE
	0	0		0	0	0	32	428	CAS	11AUG16:11:52:35
	0	0		0	0	0	16	428	CAS	11AUG16:11:52:35
	0	0		0	0	1	88	428	CAS	11AUG16:11:52:35
	0	0		0	0	1	112	428	CAS	11AUG16:11:52:35
	0	0		0	0	0	0	428	CAS	11AUG16:11:52:35
	0	0		0	0	1	80	428	CAS	11AUG16:11:52:35
	0	0		0	0	0	40	428	CAS	11AUG16:11:52:35
	0	0		0	0	1	72	428	CAS	11AUG16:11:52:35
	0	0		0	0	1	104	428	CAS	11AUG16:11:52:35
	0	0		0	0	1	96	428	CAS	11AUG16:11:52:35
DOLLAR	8	0		0	0	1	56	428	CAS	11AUG16:11:52:35
	0	0		0	0	1	120	428	CAS	11AUG16:11:52:35
	0	0		0	0	0	48	428	CAS	11AUG16:11:52:35
	0	0		0	0	1	128	428	CAS	11AUG16:11:52:35
DOLLAR	8	0		0	0	1	64	428	CAS	11AUG16:11:52:35

Obs	MODATE	DELOBS	IDXUSAGE	MEMTYPE	IDXCOUNT	PROTECT	FLAGS	COMPRESS	REUSE	SORTED
1	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
2	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
3	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
4	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
5	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
6	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
7	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
8	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
9	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
10	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
11	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
12	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
13	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
14	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.
15	11AUG16:11:52:35	0	NONE	DATA	0	---	---	NO	NO	.

SORTEDBY	CHARSET	COLLATE	NODUPKEY	NODUPREC	ENCRYPT	POINTOBS	GENMAX	GENNUM	GENNEXT	TRANSCOD
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES
.			NO	NO	NO	YES	0	.	.	YES

Examples: CONTENTS Procedure

Example 1: Describing a SAS Data Set

Features: PROC CONTENTS statement options
DATA=
OUT=

Other features: OPTIONS statement
TITLE statement

Details

This example shows the output from the CONTENTS procedure for the Cars table.

Program

```
options pagesize=40 linesize=80 nodate pageno=1;

libname mycas cas;

proc datasets library=mycas nolist;
run;

proc contents data=mycas.cars out=mycas.carsout;
   title  'The Contents of the Cars Table';
run;


proc contents data=mycas.carsout;
    title 'The Contents of the CarsOut Table';
run;
quit;
```

Program Description

Set the system options. PAGESIZE= option specifies the number of lines that compose a page of the log and output. LINESIZE= option specifies the line size for the log and for SAS procedure output. NODATE option specifies that the date and the time are not printed. PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1;
```

Set your libref.

```
libname mycas cas;
```

Specify Mycas as the procedure input library, and suppress the directory listing.

```
proc datasets library=mycas nolist;
run;
```

Create the output table CarsOut from the table Cars. Specify Cars as the table to describe and create the output table CarsOut.

```
proc contents data=mycas.cars out=mycas.carsout;
   title  'The Contents of the Cars Table';
run;
```

Display the contents of the CarsOut table.

```
proc contents data=mycas.carsout;
    title 'The Contents of the CarsOut Table';
run;
quit;
```

Output Examples

Output 2.3 *Contents of the Cars Table*

The Contents of the Cars Table

The CONTENTS Procedure

Data Set Name	MYCAS.CARS	**Observations**	428
Member Type	DATA	**Variables**	15
Engine	CAS	**Indexes**	0
Created	08/11/2016 12:33:18	**Observation Length**	160
Last Modified	08/11/2016 12:33:18	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Output 2.4 *Contents of the CarsOut Table*

The Contents of the CarsOut Table

The CONTENTS Procedure

Data Set Name	MYCAS.CARSOUT	**Observations**	15
Member Type	DATA	**Variables**	41
Engine	CAS	**Indexes**	0
Created	08/11/2016 12:33:18	**Observation Length**	928
Last Modified	08/11/2016 12:33:18	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
32	CHARSET	Char	8		Host Character Set
33	COLLATE	Char	8		Collating Sequence
28	COMPRESS	Char	8		Compression Routine
20	CRDATE	Num	8	DATETIME16.	Create Date
22	DELOBS	Num	8		Deleted Observations in Data Set
36	ENCRYPT	Char	8		Encryption Routine
19	ENGINE	Char	8		Engine Name
27	FLAGS	Char	3		Update Flags (Protect Contribute Add)
10	FORMAT	Char	32		Variable Format
12	FORMATD	Num	8		Number of Format Decimals
11	FORMATL	Num	8		Format Length
38	GENMAX	Num	8		Maximum Number of Generations
40	GENNEXT	Num	8		Next Generation Number
39	GENNUM	Num	8		Generation Number
25	IDXCOUNT	Num	8		Number of Indexes for Data Set
23	IDXUSAGE	Char	9		Use of Variable in Indexes
13	INFORMAT	Char	32		Variable Informat
15	INFORMD	Num	8		Number of Informat Decimals
14	INFORML	Num	8		Informat Length
16	JUST	Num	8		Justification
9	LABEL	Char	256		Variable Label
7	LENGTH	Num	8		Variable Length
1	LIBNAME	Char	8		Library Name
3	MEMLABEL	Char	256		Data Set Label
2	MEMNAME	Char	32		Library Member Name
24	MEMTYPE	Char	8		Library Member Type
21	MODATE	Num	8	DATETIME16.	Last Modified Date
5	NAME	Char	32		Variable Name
18	NOBS	Num	8		Observations in Data Set
34	NODUPKEY	Char	3		Sort Option: No Duplicate Keys
35	NODUPREC	Char	3		Sort Option: No Duplicate Records
17	NPOS	Num	8		Position in Buffer
37	POINTOBS	Char	3		Point to Observations
26	PROTECT	Char	3		Password Protection (Read Write Alter)
29	REUSE	Char	3		Reuse Space
30	SORTED	Num	8		Sorted and/or Validated
31	SORTEDBY	Num	8		Position of Variable in Sortedby Clause
41	TRANSCOD	Char	3		Character Variables Transcoded
6	TYPE	Num	8		Variable Type
4	TYPEMEM	Char	8		Special Data Set Type (From TYPE=)
8	VARNUM	Num	8		Variable Number

Example 2: Using the DIRECTORY Option

Features: PROC CONTENTS statement options
DATA=
DIRECTORY
OUT=

Other features: OPTIONS statement
TITLE statement

Details

This example shows the output from the CONTENTS procedure for the Cars table using the DIRECTORY option. This option prints a list of all tables that are in the specified caslib.

Program

```
options pagesize=40 linesize=80 nodate pageno=1;

libname mycas cas;


proc datasets library=mycas nolist;
run;

proc contents data=mycas.cars directory;
title 'Using the DIRECTORY Option';
run;
quit;
```

Program Description

Set the system options. The PAGESIZE= option specifies the number of lines that compose a page of the log and output. The LINESIZE= option specifies the line size for the log and for procedure output. The NODATE option specifies that the date and the time are not printed. The PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1;
```

Set your libref.

```
libname mycas cas;
```

Specify Mycas as the procedure input library, and suppress the directory listing.

```
proc datasets library=mycas nolist;
run;
```

Specify Cars as the table to describe. Use the DIRECTORY option to print a listing of all the tables that are in the Mycas caslib.

```
proc contents data=mycas.cars directory;
```

```
title 'Using the DIRECTORY Option';
run;
quit;
```

Output Examples

Output 2.5 *Using the DIRECTORY Option - Section 1*

Using the DIRECTORY Option

The CONTENTS Procedure

Directory	
Libref	MYCAS
Engine	CAS
Physical Name	ddd7fa0c
Server Session UUID	ddd7fa0c
Server Session Name	CASAUTO
Server Host	10.21.15.152
Server Session Port	5570
Server Session CASLIB	CASUSER

#	Name	Member Type	Number of Rows	Number of Columns	Last Modified	Data Encoding	Scope
1	CARS	DATA	428	15	04/18/2016 11:34:09	utf-8	Session
2	LASTMONTH	DATA	12	7	04/18/2016 11:33:32	utf-8	Session

Using the DIRECTORY Option

The CONTENTS Procedure

Data Set Name	MYCAS.CARS	Observations	428
Member Type	DATA	Variables	15
Engine	CAS	Indexes	0
Created	08/11/2016 10:56:31	Observation Length	160
Last Modified	08/11/2016 10:56:31	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Example 3: Using the DIRECTORY and DETAILS Options

Features: PROC CONTENTS statement options
- DATA=
- DETAILS
- DIRECTORY
- OUT=

Other features: OPTIONS statement
TITLE statement

Details

This example shows the output from the CONTENTS procedure for the Group table using the DIRECTORY option. This option prints a list of all SAS files that are in the

specified SAS library. The DETAILS option includes information in the output about the number of rows, number of variables, and table labels.

Program

```
options pagesize=40 linesize=80 nodate pageno=1;

libname mycas cas;

proc datasets library=mycas nolist;
run;

proc contents data=macas.cars directory details;
title 'Using the DIRECTORY and DETAILS Options';
run;
```

Program Description

Set the system options. The PAGESIZE= option specifies the number of lines that compose a page of the log and output. The LINESIZE= option specifies the line size for the log and for SAS procedure output. The NODATE option specifies that the date and the time are not printed. The PAGENO= option specifies a beginning page number for the next page of output.

```
options pagesize=40 linesize=80 nodate pageno=1;
```

Set your libref.

```
libname mycas cas;
```

Specify Mycas as the procedure input library, and suppress the directory listing.

```
proc datasets library=mycas nolist;
run;
```

Specify Cars as the table. Use the DIRECTORY option to print a listing of all the tables that are in the Mycas caslib. Use the DETAILS options for additional columns of information in the Cars output.

```
proc contents data=macas.cars directory details;
title 'Using the DIRECTORY and DETAILS Options';
run;
```

Output Examples

Output 2.6 *Using the DIRECTORY and DETAILS Options*

Using the DIRECTORY and DETAILS Options

The CONTENTS Procedure

Directory	
Libref	MYCAS
Engine	CAS
Physical Name	7f498295-
Server Session UUID	7f498295
Server Session Name	CASAUTO
Server Host	10.21.15.152
Server Session Port	5570
Server Session CASLIB	CASUSER

#	Name	Member Type	Obs, Entries or Indexes	Vars	Label	Number of Rows	Number of Columns	Last Modified	Data Encoding	Scope
1	CARS	DATA	428	15		428	15	04/15/2016 16:44:54	utf-8	Session
2	LASTMONTH	DATA	12	7		12	7	04/15/2016 16:24:24	utf-8	Session

Using the DIRECTORY and DETAILS Options

The CONTENTS Procedure

Data Set Name	MYCAS.CARS	Observations	428
Member Type	DATA	Variables	15
Engine	CAS	Indexes	0
Created	03/31/2016 15:31:25	Observation Length	160
Last Modified	03/31/2016 15:31:25	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Chapter 3

COPY Procedure

Overview: COPY Procedure

The COPY procedure copies one or more tables to and from a library.

Concepts

Using the COPY Procedure Instead of the COPY Statement

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. Here is a list of differences:

- The IN= argument is required with PROC COPY. In the COPY statement, IN= is optional. If omitted, the default value is the libref of the procedure input library.
- The COPY statement honors the NOWARN option but PROC COPY does not.

Copying Character Values

When you copy a SAS data set with a CHAR value to a CAS engine table, CHAR is copied as characters.

When you copy a CAS table to a SAS data set, CHAR and VARCHAR are copied as characters.

When you copy a CAS table to another CAS table, VARCHAR copies as VARCHAR.

Copying a CAS Table to Another CAS Table

The COPY procedure can copy a CAS table to another CAS table if the initial table is small enough. All CAS tables are in-memory. However, the performance is very slow. The reason for the performance issue is that all the data needs to go into the client before it can go into another in-memory table.

Note: **Small enough** means that the size of the data is less than or equal to the setting of the ReadTransferSize option.

The most efficient way to copy a CAS table into another CAS table is to use the CASUTIL procedure. Use the CASUTIL procedure to load a new in-memory table from the original data source, or persist the in-memory table to disk to be loaded as a new in-memory table. There are a few extra steps to copy from an in-memory table to another in-memory table. For more information about the CASUTIL procedure, see *SAS Cloud Analytic Services: Language Reference*.

Copying an Entire Library

To copy an entire CAS library, simply specify an input library and an output library following the COPY procedure. For example, the following statements copy all the tables in the Source library into the Dest library:

```
proc copy in=source out=dest;
run;
```

Copying Selected Tables

To copy selected CAS tables, use a SELECT or EXCLUDE statement.

You can also select or exclude an abbreviated list of tables. For example, the following statement selects tables Tabs, Test1, Test2, and Test3:

```
select tabs test1-test3;
```

Also, you can select a group of tables whose names begin with the same letter or letters by entering the common letters followed by a colon (:). For example, you can select the four tables in the previous example and all other tables having names that begin with the letter T by specifying the following statement:

```
select t:;
```

You specify tables to exclude in the same way that you specify those to select. That is, you can list individual table names, use an abbreviated list, or specify a common letter or letters followed by a colon (:). For example, the following statement excludes the tables Stats, Teams1, Teams2, Teams3, Teams4 and all the tables that begin with the letters RBI from the copy operation:

```
exclude stats teams1-teams4 rbi:;
```

Copying Compressed Tables

When copying previously compressed tables, the following occurs:

- if a SAS data set is compressed, then it retains the COMPRESS=YES value on a CAS table.
- if a CAS table is compressed, then it converts to a SAS data set with the COMPRESS=CHAR value.

Syntax: COPY Procedure

Restrictions: PROC COPY does not support SAS data set options.

When using PROC COPY with the CAS engine, if the input table and output table are in the same CAS session, only one million rows are supported.

PROC COPY
OUT=*libref-1*
<CLONE | NOCLONE>
IN=*libref-2*
<SELECT>
<EXCLUDE>
<MEMTYPE=(*member-type(s)*)>
<MOVE >
<OVERRIDE=(*ds-option-1=value-1 <ds-option-2=value-2 …>*) >;

Statement	Task	Example
PROC COPY	Copy one or more tables in a library	Ex. 1

PROC COPY Statement

Copies all or some of the tables in a CAS library.

Syntax

PROC COPY OUT= *<option(s)>*
IN= *<option(s)>*;

Required Arguments

OUT=*libref-1*
names the library to copy tables to.

Alias OUTLIB= and OUTDD=

IN=*libref-2*
names the CAS engine library containing tables to copy.

Alias	INLIB= and INDD=
Default	the libref of the procedure input
Interaction	To copy only selected tables, use the SELECT or EXCLUDE statements.

Optional Arguments

CLONE | NOCLONE
specifies whether to copy the following attributes. The only attribute that applies to the CAS engine is COMPRESS.

The following table summarizes how the COPY statement works:

Table 3.1 *CLONE Interaction with Attributes*

Attribute	To	CLONE or NOCLONE	Description
BUFSIZE= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	OVERRIDE=(BUFSIZE=*value other than default*)
		NOCLONE	Uses setting of BUFSIZE= system option
	CAS table to CAS table		
COMPRESS= SAS data set - COMPRESS=BINARY | CHAR | NO CAS table - COMPRESS=NO | YES	SAS data set to CAS table	CLONE	A compressed SAS data set becomes a compressed CAS table unless OVERRIDE= is used. An uncompressed SAS data set becomes an uncompressed CAS table unless OVERRIDE= is used.
		NOCLONE	Follows the CAS LIBNAME setting.
	CAS table to SAS data set	CLONE	A compressed CAS table becomes a SAS data set CHAR value variable unless OVERRIDE= is used.
		NOCLONE	The COMPRESS= system option or LIBNAME option value is used.

Attribute	To	CLONE or NOCLONE	Description
	CAS table to CAS table	CLONE	Keeps the current setting.
		NOCLONE	Set using the CAS LIBNAME setting for the OUT= libref.
REUSE= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	REUSE=NO unless the OVERRIDE= or REUSE=YES system options are used.
		NOCLONE	Uses the REUSE= system option value.
	CAS table to CAS table		
POINTOBS= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	POINTOBS=NO unless OVERRIDE= is used.
		NOCLONE	POINTOBS=NO, if the CAS table is compressed and the LIBNAME statement has POINTOBS=NO. POINTOBS=YES, if the CAS table is compressed and the LIBNAME option is missing.
	CAS table to CAS table		
OUTREP= CAS engine does not support.	SAS data set to CAS table	CLONE	Converts to LINUX_86_64 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)
		NOCLONE	Converts to LINUX_86_64 if needed.

Attribute	To	CLONE or NOCLONE	Description
	CAS table to SAS data set		
		CLONE	Keeps data representation. (A warning is sent to the log if the OVERRIDE= option is used.)
	CAS table to CAS table		
ENCODING= CAS engine supports UTF-8 only. Changing encoding is not implemented for the CAS engine.	SAS data set to CAS table	CLONE	Converts to UTF-8 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)
		NOCLONE	Converts to UTF-8 if needed.
	CAS table to SAS data set	CLONE	Keeps the UTF-8 encoding unless OVERRIDE= is used.
		NOCLONE	Keeps the UTF-8 encoding unless OUTENCODING= is used in the output SAS data set LIBNAME is used.
	CAS table to CAS table		

EXCLUDE

Excludes tables from copying.

Restrictions The EXCLUDE statement must follow a COPY statement

The EXCLUDE statement cannot appear in the same COPY step with a SELECT statement

MEMTYPE=(*member-type(s)*)

restricts processing to one or more member types. Member types available for the CAS engine are DATA and ALL.

Alias MT=, MTYPE=

Default If you omit MEMTYPE= in the PROC COPY, the default is MEMTYPE=ALL.

MOVE
moves tables from the input library (named with the IN= option) to the output library (named with the OUT= option). And deletes the original tables from the input library.

Restriction The MOVE option can be used to delete a table in a CAS library only if the IN= engine supports the deletion of tables.

OVERRIDE=(*ds-option-1=value-1 <ds-option-2=value-2> ...*)
overrides specified output table options copied from the input table. Some table options might not be appropriate in the output table context of COPY.

Restriction The OVERRIDE option is ignored if the NOCLONE option is specified. However, it can be used to modify attributes other than those controlled by the NOCLONE option.

SELECT
Selects tables for copying.

Restrictions The SELECT statement must follow a COPY statement

The SELECT statement cannot appear with an EXCLUDE statement in the same COPY step

Using the COPY Procedure

Compressing Output Tables

When copying previously compressed tables, the following occurs:

- if a SAS data set is compressed, then it retains the COMPRESS=YES value on the CAS table.
- if a CAS table is compressed, then it converts to a SAS data set with the COMPRESS=CHAR value.

The COPY procedure does not support SAS data set options. Therefore, you cannot use the COMPRESS= SAS data set option in PROC COPY or a COPY statement from PROC DATASETS. To compress an OUTPUT table generated by PROC COPY, you can use the COMPRESS=YES system option before the PROC COPY statement with the NOCLONE option.

Example: Copy a SAS Data Set to a CAS Table

Features: PROC COPY statement options
IN=
OUT=
SELECT statement

Details

This example demonstrates how to copy a SAS data set into a CAS engine table.

Program

```
libname mycas cas;
libname mylib 'BASE-engine-library';

proc copy in=mylib out=mycas;
   select monthly;
run;
quit;
```

Program Description

Assign library references. Select the data set that you want to copy into a CAS table.

```
libname mycas cas;
libname mylib 'BASE-engine-library';
```

Use PROC COPY and the SELECT statement. Copy a SAS data set into a CAS table.

```
proc copy in=mylib out=mycas;
   select monthly;
run;
quit;
```

Log Examples

Log 3.1 *MyLib Library Log*

```
57         libname mycas cas;
 NOTE: Libref MYCAS was successfully assigned as follows:
       Engine:        CAS
       Physical Name: 1f436ced
 58         libname mylib 'BASE-engine-library';
 NOTE: Libref MYLIB was successfully assigned as follows:
       Engine:        V9
       Physical Name: BASE-engine-library
 59
 60         proc copy in=mylib out=mycas;
 61            select monthly;
 62         run;

 NOTE: Copying MYLIB.MONTHLY to MYCAS.MONTHLY (memtype=DATA).
 NOTE: There were 12012 observations read from the data set MYLIB.MONTHLY.
 NOTE: The data set MYCAS.MONTHLY has 12012 observations and 7 variables.
 NOTE: PROCEDURE COPY used (Total process time):
       real time           0.06 seconds
       cpu time            0.02 seconds
```

Chapter 4

DATASETS Procedure

Overview: DATASETS Procedure

Managing Tables Using the DATASETS Procedure

Note: This chapter is specific to CAS engine tables in SAS Viya. For more information about CAS tables and the CAS engine, see *SAS Cloud Analytic Services: Language Reference*.

The DATASETS procedure is a utility procedure that can manage your tables. With PROC DATASETS, you can do the following if the LIB= option is a caslib:

- copy tables from one caslib to another
- delete tables
- list the tables that are contained in a caslib

- list the attributes of a table:
  - the date on which the data was last modified
  - whether the data is compressed
- append tables

Notes

- The term *member* often appears as a synonym for *table*.
- You cannot change the length of a variable using the LENGTH statement or the LENGTH= option in an ATTRIB statement.
- If you have a caslib containing a large number of tables, the DATASETS procedure might show an increase in process time. You might want to reorganize your caslib into smaller libraries for better performance.

Concepts: DATASETS Procedure

Procedure Execution

Execution of Statements

When you start the DATASETS procedure, you specify the procedure input caslib in the PROC DATASETS statement. If you omit a procedure input caslib, the procedure processes the current default caslib. To specify a new procedure input caslib, specify the LIB= option with a CAS libref.

Statements execute in the order in which they are written. Use CONTENTS, COPY, CONTENTS if you want to see the contents of a table, copy a table, and then visually compare the contents of the second table with the first.

Ending the Procedure

To stop the DATASETS procedure, you must issue a QUIT statement, a new PROC statement, or a DATA statement. Submitting a QUIT statement executes any statements that have not executed.

Syntax: DATASETS Procedure

```
PROC DATASETS <option(s)>;
   APPEND BASE=<libref.>SAS-data-set
     <DATA=<libref.>data-set>
     <FORCE>
     <NOWARN>;
   CONTENTS <option(s)>;
   COPY OUT=libref-1
     <CLONE | NOCLONE>
```

IN=*libref-2*
<MOVE >
<FORCE>
<MEMTYPE=*member-type*>
<OVERRIDE=(*ds-option-1=value-1 <ds-option-2=value-2 …>*) >;
<SELECT *table-name(s);*>
<n0jsyb7b22aux8n1od51m5w9xeruEXCLUDE *table-name(s)*;>
DELETE *table-name(s)*;
SAVE *table-name(s)*;

Statement	Task
PROC DATASETS	Manage tables
APPEND	Add rows from one table or data set to the end of another data set
CONTENTS	Describe the contents of one or more tables and prints a directory of the caslib
COPY	Copy all or some of the tables
DELETE	Delete tables
EXCLUDE	Exclude tables from copying
SAVE	Delete all the tables except the ones listed in the SAVE statement
SELECT	Select tables for copying

PROC DATASETS Statement

Manages tables.

Syntax

PROC DATASETS <*option(s)*>;

Summary of Optional Arguments

DETAILS | NODETAILS
: includes information in the log about the number of rows, number of variables, and table labels.

FORCE
: forces an Append operation.

KILL
: deletes tables.

LIBRARY=*libref*
: specifies the procedure input/output caslib.

NODETAILS
: see the description of DETAILS | NODETAILS.

NOLIST
: suppresses the printing of the directory.

NOPRINT
: suppresses the printing of the output to the log and listing.

NOWARN
: suppresses some error processing.

Optional Arguments

DETAILS | NODETAILS

determines whether the following columns are written to the log:

Obs or Entries
: gives the number of rows for tables.

Vars
: gives the number of variables for DATA type. If the number of variables cannot be determined in the table, the value in this column is set to missing.

Label
: contains the label associated with the table.

Default If neither DETAILS or NODETAILS is specified, the default is the system option setting. The default system option setting is NODETAILS.

FORCE

Forces all APPEND statements to concatenate two data sets even when the variables in the data sets are not exactly the same. The APPEND statement drops the extra variables and issues a warning message to the log unless the NOWARN option is specified (either with the APPEND statement or PROC DATASETS).

KILL

deletes all tables in the caslib that are available for processing. The following example deletes all the data tables in the OldSales library:.

```
proc datasets lib=oldsales  kill; quit;
```

CAUTION:
The KILL option deletes the tables immediately after you submit the statement.

LIBRARY=*libref*

names the caslib that the procedure processes. This caslib is the *procedure input/output caslib*.

Alias LIB=

NODETAILS

NOLIST

suppresses the printing of the directory of the tables in the log and any open non-LISTING destination.

NOPRINT

suppresses the printing of the output and the printing of the directory of the tables in the log and any open non-LISTING destination. The NOPRINT option is a combination of the NOLIST option and the NOPRINT option in the CONTENTS statement.

NOWARN
suppresses the error processing that occurs when a table that is specified in a SAVE, DELETE, or COPY statement is not in the procedure input caslib.

APPEND Statement

Adds rows from a SAS data set or CAS table to the end of a SAS data set.

Restrictions: The BASE= option and OUT= option cannot be a CAS table.
You cannot concatenate two CAS tables or a SAS data set to a CAS table.

Requirement: The BASE= SAS data set must be a member of a library that supports update processing.

Tip: You can specify most data set options for the BASE= option and DATA= option. However, if you specify DROP=, KEEP=, or RENAME= data set option for the BASE= SAS data set, the option is ignored. You can use any global statements as well.

See: "Example: Concatenating a CAS Table to a SAS Data Set" on page 8

Syntax

APPEND BASE=*<libref.>SAS-data-set*
<DATA=*<libref.>data-set*>
<FORCE>
<NOWARN>;

Required Argument

BASE=*<libref.> SAS-data-set*
names the SAS data set to which you want to add rows. BASE= option cannot specify a CAS table.

libref
specifies the library that contains the SAS data set. If you omit the libref, the default is the libref for the procedure input library. If you are using PROC APPEND, the default for libref is either Work or User.

data-set
names a SAS data set. If the APPEND statement cannot find an existing data set with this name, it creates a new data set in the library. That is, you can use the APPEND statement to create a data set by specifying a new data set name in the BASE= argument.

Whether you are creating a new data set or appending to an existing data set, the BASE= SAS data set is the current data set after all Append operations.

Alias OUT=

Optional Arguments

DATA=*<libref.> data-set*
names the SAS data set or CAS table containing rows that you want to append to the end of the data set specified in the BASE= argument.

libref
specifies the library that contains the data set. If you omit libref, the default is the libref for the procedure input library. The DATA= data set or table can be from any library. You must use the two-level name if the data set resides in a library other than the procedure input library.

data-set
names a data set or table. If the APPEND statement cannot find an existing data set with this name, it stops processing.

Alias	NEW=
Default	the most recently created data set, from any library

FORCE
forces the APPEND statement to concatenate SAS data sets or a CAS table to a SAS data set when the DATA= data set contains variables that meet one of the following criteria:

- are not in the BASE= SAS data set
- do not have the same type as the variables in the BASE= SAS data set
- are longer than the variables in the BASE= SAS data set

NOWARN
suppresses the warning when used with the FORCE option to concatenate two SAS data sets or a CAS table and a SAS data set with different variables.

Details

Using the APPEND Procedure Instead of the APPEND Statement

The only difference between the APPEND procedure and the APPEND statement in PROC DATASETS, is the default for libref in the BASE= and DATA= arguments. For PROC APPEND, the default is either Work or User. For the APPEND statement, the default is the libref of the procedure input library.

For more information, see Chapter 1, “APPEND Procedure,” on page 3.

Appending with the CAS Engine

Using the APPEND statement with the CAS engine, you can do the following:

- add rows from a CAS table to the end of a SAS data set.
- add rows from a SAS data set to the end of another SAS data set.

Using the APPEND statement, you cannot do the following:

- add rows from a CAS table to the end of another CAS table.
- add rows from a SAS data set to the end of a CAS table.

The BASE= option or OUT= option cannot be a CAS table.

Using the Block I/O Method to Append

Note: The block I/O method cannot be used when appending a CAS table to a SAS data set.

The block I/O method is used to append blocks of data instead of one row at a time. This method increases performance when you are appending large data sets. SAS determines

whether to use the block I/O method. Not all tables can use the block I/O method. There are restrictions set by the APPEND statement and the BASE engine.

To display information in the log about the append method that is being used, you can specify the MSGLEVEL= system option as follows:

```
options msglevel=i;
```

The following message is written to the log, if the block I/O method is not used:

```
INFO: Data set block I/O cannot be used because:
```

If the APPEND statement determines that the block I/O will not be used, one of the following explanations is written to the log:

```
INFO: - The data sets use different engines, have different
variables or have attributes that might differ.
```

```
INFO: - There is a WHERE clause present.
```

```
INFO: - There is no member level locking.
```

If the BASE engine determines that the block I/O method will not be used, one of the following explanations is written to the log:

```
INFO: - Cross Environment Data Access is being used.
```

```
INFO: - The table is compressed.
```

Appending to a Compressed Data Set

You can concatenate compressed SAS data sets. Either or both of the BASE= SAS data set and DATA= data set or CAS table can be compressed. If the BASE= data set allows the reuse of space from deleted rows, the APPEND statement might insert the rows into the middle of the BASE= data set.

For information about the COMPRESS= and REUSE= data set and system options, see *SAS Viya Data Set Options: Reference* and *SAS Viya System Options: Reference*.

Appending to Data Sets That Contain Variables with Different Attributes

If a variable has different attributes in the BASE= SAS data set than it does in the DATA= data set or table, the attributes in the BASE= data set prevail.

If the formats in the DATA= SAS data set or CAS table are different from those in the BASE= SAS data set, then the formats in the BASE= data set are used. However, the data from the DATA= data set or table is not converted in order to be consistent with the formats in the BASE= data set. The result could be data that seems to be incorrect. A warning message is displayed in the log.

Use the FORCE option if one of the following occurs:

- if the length of a variable is longer in the DATA= SAS data set or CAS table than in the BASE= data set
- if the same variable is a character variable in one data set or table and a numeric variable in the other

Using FORCE has the following consequences:

- The length of the variables in the BASE= SAS data set takes precedence. The values might be truncated from the DATA= data set or CAS table to fit them into the length that is specified in the BASE= data set.

- The type of the variables in the BASE= data set takes precedence. The APPEND statement replaces values of the wrong type (all values for the variable in the DATA= data set or table) with missing values.

System Failures

If a system failure or some other type of interruption occurs while the procedure is executing, the Append operation might not be successful; it is possible that not all, perhaps none, of the rows are added to the BASE= SAS data set. In addition, the BASE= data set might suffer damage. The Append operation performs an update in place, which means that it does not make a copy of the original data set before it begins to append rows.

CONTENTS Statement

Describes the contents of one or more CAS tables and prints the directory of the library.

Restriction: You cannot use the WHERE option to affect the output because PROC CONTENTS does not process any rows.

Tips: You can use data set options with the DATA= option and the OUT= option.

The ORDER= option does not affect the order of the OUT= table.

Syntax

CONTENTS <*option(s)*>;

Optional Arguments

DATA=*CAS-table-specification*

specifies an entire caslib or a specific table within a library. *CAS-table-specification* can take one of the following forms:

<libref.>CAS-table

names one table to process. The default for libref is the libref of the procedure input library. For example, to obtain the contents of the table HtWt from the procedure input library, use the following CONTENTS statement:

```
contents data=HtWt;
```

***<libref.>*_ALL_**

gives you information about all CAS tables.*libref* refers to the caslib. The default for libref is the libref of the procedure input library. DATA=_ALL_ automatically prints a listing of the tables that are contained in the caslib.

Default most recently created table in your job or session, from any library.

DETAILS | NODETAILS

includes information in the output about the number of rows, number of variables, and table labels. DETAILS includes additional columns of information in the output, but only if DIRECTORY is also specified.

Default If neither DETAILS nor NODETAILS is specified, the defaults are as follows: for the CONTENTS procedure, the default is the system option setting, which is NODETAILS; for the CONTENTS statement, the default

is whatever is specified in the PROC DATASETS statement, which also defaults to the system option setting.

DIRECTORY
prints a list of all CAS tables in the specified library. If DETAILS is also specified, using DIRECTORY causes the additional columns to be included.

FMTLEN
prints the length of the informat or format. You must specify a length for the informat or format when you associate it with a variable. If you do not specify the length, it does not appear in the output of the CONTENTS statement. You must use the FMTLEN option for the length to appear. The length also appears in the FORMATL or INFORML variable in the output table.

NODS
suppresses printing the contents of individual CAS tables when you specify _ALL_ in the DATA= option. The CONTENTS statement prints only the library directory. You cannot use the NODS option when you specify only one table in the DATA= option.

NOPRINT
suppresses printing the output of the CONTENTS statement.

ORDER=COLLATE | CASECOLLATE | IGNORECASE | VARNUM

COLLATE
prints a list of variables in alphabetical order beginning with uppercase and then lowercase names.

CASECOLLATE
prints a list of variables in alphabetical order even if they include mixed-case names and numerics.

IGNORECASE
prints a list of variables in alphabetical order ignoring the case of the letters.

VARNUM
is the same as the VARNUM option.

Note The ORDER= option does not affect the order of the OUT= table.

OUT=*table-name*
names an output table.

Tip OUT= does not suppress the printed output from the statement. If you want to suppress the printed output, you must use the NOPRINT option.

SHORT
prints only the list of variable names for the table.

Restriction If the list of variables is more than 32,767 characters, the list is truncated and a WARNING is written to the log. To get a complete list of the variables, request an alphabetical listing of the variables.

VARNUM
prints a list of the variable names in the order of their logical position in the table. The physical position of the variable in the table is engine-dependent.

Details

Using the CONTENTS Procedure Instead of the CONTENTS Statement

The only difference between the CONTENTS procedure and the CONTENTS statement in PROC DATASETS is the default for libref in the DATA= option. For PROC CONTENTS, the default is Work. For the CONTENTS statement, the default is the libref of the procedure input library.

For more information, see Chapter 2, “CONTENTS Procedure,” on page 13.

Contents of a CAS Table and Library

The CONTENTS statement shows the contents of a table and prints the directory of the caslib.

```
proc datasets lib=mycas;
contents data=cars;
run;
```

Output 4.1 *Mycas Caslib and Mycas.Cars Table*

Directory	
Libref	MYCAS
Engine	CAS
Physical Name	ddd7fa0c
Server Session UUID	ddd7fa0c
Server Session Name	CASAUTO
Server Session Port	5570
Server Session CASLIB	CASUSER

#	Name	Member Type	Number of Rows	Number of Columns	Last Modified	Data Encoding	Scope
1	CARS	DATA	428	15	04/18/2016 11:44:24	utf-8	Session
2	CARSOUT	DATA	15	41	04/18/2016 11:44:24	utf-8	Session
3	LASTMONTH	DATA	12	7	04/18/2016 11:33:32	utf-8	Session

Data Set Name	MYCAS.CARS	Observations	428
Member Type	DATA	Variables	15
Engine	CAS	Indexes	0
Created	08/11/2016 10:49:36	Observation Length	136
Last Modified	08/11/2016 10:49:36	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
14	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
15	Horsepower	Num	8		
9	Length	Num	8		Length (IN)
10	MPG_City	Num	8		MPG (City)
11	MPG_Highway	Num	8		MPG (Highway)
7	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Varchar	40		
4	Origin	Char	6		
3	Type	Char	8		
12	Weight	Num	8		Weight (LBS)
13	Wheelbase	Num	8		Wheelbase (IN)
6	invoice	Num	8	DOLLAR8.	

COPY Statement

Copies all or some of the tables in a library.

Restriction: The COPY statement does not support table options.

Syntax

COPY OUT=*libref-1*
<CLONE | NOCLONE>
IN=*libref-2*
<MOVE >
<OVERRIDE=(*ds-option-1=value-1 <ds-option-2=value-2 …>*) >;
<SELECT *table-name(s)*>;
<EXCLUDE *table-name(s)*;>
<MEMTYPE=(*member-type(s)*)>

Required Argument

OUT=*libref-1*
names the library to copy CAS tables to.

Alias OUTLIB= and OUTDD=

Optional Arguments

CLONE | NOCLONE
specifies whether to copy data set attributes. The only attribute that can be used with the CAS engine is COMPRESS.

Attributes are specified with data set options, system options, or LIBNAME statement options. The CAS engine supports only the COMPRESS=YES | NO option. No other attributes are supported by the CAS engine.

The following CAS table summarizes how the COPY statement works:

Table 4.1 *CLONE Interaction with Attributes*

Attribute	To	CLONE or NOCLONE	Description
BUFSIZE= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	OVERRIDE=(BUFSIZE=*value other than default*)
		NOCLONE	Uses setting of BUFSIZE= system option
	CAS table to CAS table		
COMPRESS= SAS data set - COMPRESS=BINARY | CHAR | NO CAS table - COMPRESS=NO | YES	SAS data set to CAS table	CLONE	A compressed SAS data set becomes a compressed CAS table unless OVERRIDE= is used. An uncompressed SAS data set becomes an uncompressed CAS table unless OVERRIDE= is used.
		NOCLONE	Follows the CAS LIBNAME setting.
	CAS table to SAS data set	CLONE	Compressed CAS table becomes a SAS data set CHAR value variable unless the OVERRIDE= is used.

Attribute	To	CLONE or NOCLONE	Description
		NOCLONE	The COMPRESS= system option or LIBNAME option value is used.
	CAS table to CAS table	CLONE	Keeps the current setting.
		NOCLONE	Set using the CAS LIBNAME setting for the OUT= libref.
REUSE= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	REUSE=NO unless the OVERRIDE= or REUSE=YES system options are used.
		NOCLONE	Uses the REUSE= system option value.
	CAS table to CAS table		
POINTOBS= CAS engine does not support.	SAS data set to CAS table		
	CAS table to SAS data set	CLONE	POINTOBS=NO unless OVERRIDE= is used.
		NOCLONE	POINTOBS=NO, if the CAS table is compressed and the LIBNAME statement has POINTOBS=NO. POINTOBS=YES, if the CAS table is compressed and the LIBNAME option is missing.
	CAS table to CAS table		
OUTREP= CAS engine does not support.	SAS data set to CAS table	CLONE	Converts to LINUX_86_64 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)

Attribute	To	CLONE or NOCLONE	Description
		NOCLONE	Converts to LINUX_86_64 if needed.
	CAS table to SAS data set		
		CLONE	Keeps data representation. (A warning is sent to the log if the OVERRIDE= option is used.)
	CAS table to CAS table		
ENCODING= CAS engine supports UTF-8 only. Changing encoding is not implemented for the CAS engine.	SAS data set to CAS table	CLONE	Converts to UTF-8 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)
		NOCLONE	Converts to UTF-8 if needed.
	CAS table to SAS data set	CLONE	Keeps the UTF-8 encoding unless OVERRIDE= is used.
		NOCLONE	Keeps the UTF-8 encoding unless OUTENCODING= is used in the output data set LIBNAME is used.
	CAS table to CAS table		

IN=*libref-2*
names the library containing CAS tables to copy.

Alias INLIB= and INDD=

Default the libref of the procedure input library

Interaction To copy only selected CAS tables, use the SELECT or EXCLUDE statements.

MEMTYPE=(*member-type(s)*)
restricts processing to one or more member types. Member types available for the CAS engine are DATA and ALL.

Alias MT=, MTYPE=

Default If you omit MEMTYPE= in the PROC DATASETS statement, the default is MEMTYPE=ALL.

Tip To limit the size of the library, use the MEMTYPE= option for the SAS data set.

MOVE
moves CAS tables from the input library (named with the IN= option) to the output library (named with the OUT= option). And deletes the original CAS tables from the input library.

Restriction The MOVE option can be used to delete a CAS table in a library only if the IN= engine supports the deletion of CAS tables.

OVERRIDE=(*ds-option-1=value-1 <ds-option-2=value-2> ...*)
overrides specified output SAS data set options copied from the input CAS table. Some SAS data set options might not be appropriate in the output CAS table context of COPY.

Restriction The OVERRIDE= option is ignored if the NOCLONE option is specified. However, it can be used to modify attributes other than those controlled by the NOCLONE option.

Details

Using the COPY Procedure Instead of the COPY Statement

Generally, the COPY procedure functions the same as the COPY statement in the DATASETS procedure. Here is a list of differences:

- The IN= argument is required with PROC COPY. In the COPY statement, IN= is optional. If omitted, the default value is the libref of the procedure input library.
- The COPY statement honors the NOWARN option but PROC COPY does not.

For more information, see Chapter 3, “COPY Procedure,” on page 33.

Copying a CAS Table to Another CAS Table

The COPY procedure can copy a CAS table to another CAS table if the initial table is small enough. All CAS tables are in-memory. However, the performance is very slow. The reason for the performance issue is that all the data needs to go into the client before it can go into another in-memory table.

Note: **Small enough** means that the size of the data is less than or equal to the setting of the ReadTransferSize option.

The most efficient way to copy a CAS table into another CAS table is to use the CASUTIL procedure. Use the CASUTIL procedure to load a new in-memory table from the original data source, or persist the in-memory table to disk to be loaded as a new in-memory table. There are a few extra steps to copy from an in-memory table to another in-memory table. For more information about the CASUTIL procedure, see *SAS Cloud Analytic Services: Language Reference*.

Copying an Entire Library

To copy an entire caslib, simply specify an input library and an output library following the COPY procedure. For example, the following statements copy all the tables in the Source library into the Dest library:

```
proc datasets library=source;
   copy out=dest;
run;
```

Copying Selected Tables

To copy selected tables, use a SELECT or EXCLUDE statement.

You can also select or exclude an abbreviated list of members. For example, the following statement selects members Tabs, Test1, Test2, and Test3:

```
select tabs test1-test3;
```

Also, you can select a group of tables whose names begin with the same letter or letters by entering the common letters followed by a colon (:). For example, you can select the four tables in the previous example and all other tables having names that begin with the letter T by specifying the following statement:

```
select t:;
```

You specify tables to exclude in the same way that you specify those to select. That is, you can list individual tables names, use an abbreviated list, or specify a common letter or letters followed by a colon (:). For example, the following statement excludes the tables Stats, Teams1, Teams2, Teams3, Teams4 and all the tables that begin with the letters RBI from the copy operation:

```
exclude stats teams1-teams4 rbi:;
```

Copying Compressed Tables

When copying previously compressed tables, the following occurs:

- if a SAS data set is compressed, then it retains the COMPRESS=YES value on a CAS table.
- if a CAS table is compressed, then it converts to a SAS data set with the COMPRESS=CHAR value.

DELETE Statement

Deletes tables from a library.

Syntax

DELETE *table-name(s)* ;

Required Argument

table-name(s)
specifies one or more tables that you want to delete. You can also use a numbered range list or colon list.

Details

The Basics

If you attempt to delete a table that does not exist in the procedure input caslib, PROC DATASETS issues a message and continues processing. If NOWARN is used, no message is issued.

Note: MEMTYPE= does not apply to CAS tables

EXCLUDE Statement

Excludes tables from copying.

Restrictions: The EXCLUDE statement must follow a COPY statement

The EXCLUDE statement cannot appear in the same COPY step with a SELECT statement

Syntax

EXCLUDE *table-name(s)* </ MEMTYPE=*member-type*>;

Required Argument

table-name(s)

specifies one or more tables to exclude from the copy operation. All tables you name in the EXCLUDE statement must be in the library that is specified in the IN= option in the COPY statement.

You can use the following shortcuts to list several tables in the EXCLUDE statement:

Notation	Meaning
x1–xn	Specifies tables X1 through Xn. The numbers must be consecutive.
x:	Specifies all tables that begin with the letter X.

Optional Argument

MEMTYPE=*member-type*

restricts processing to one member type. You can use the option either in parentheses after the name of each table or after a forward slash.

Alias	MTYPE=, MT=
Default	If you do not specify MEMTYPE= in the PROC DATASETS statement, the COPY statement, or in the EXCLUDE statement, the default is MEMTYPE=ALL.
Note	MEMTYPE= does not apply to CAS tables.

Details

Excluding Several Like-Named Tables

You can use shortcuts for listing several tables in the EXCLUDE statement.

SAVE Statement

Deletes all the tables in a library except the ones listed in the SAVE statement.

Syntax

SAVE *table-name(s)* ;

Required Argument

table-name(s)
specifies one or more tables that you do not want to delete from the library.

SELECT Statement

Selects tables for copying.

Restrictions: The SELECT statement must follow a COPY statement

The SELECT statement cannot appear with an EXCLUDE statement in the same COPY step

Syntax

SELECT *table-name(s)*; </ <MEMTYPE=*member-type*>>;

Required Argument

table-name(s)
specifies one or more tables that you want to copy . All of the tables that you name must be in the data library that is referenced by the libref named in the IN= option in the COPY statement.

Optional Argument

MEMTYPE=*member-type*
restricts processing to one member type in a library. You can use the option after a forward slash. The MEMTYPE= values available for the CAS engine are DATA and ALL.

Alias MTYPE= and MT=

Default If you do not specify the MEMTYPE= option in the PROC DATASETS statement, in the COPY statement, or in the SELECT statement, the default is MEMTYPE=ALL.

Details

Selecting Several Like-Named Tables

You can use shortcuts for listing several tables in the SELECT statement:

Notation	Meaning
x1–xn	Specifies tables X1 through Xn. The numbers must be consecutive.
x:	Specifies all tables that begin with the letter X.

Results: DATASETS Procedure

Directory Listing to the Log

The PROC DATASETS statement lists the tables in the procedure input library unless the NOLIST option is specified. The NOLIST option prevents the creation of the procedure results that go to the log. If you specify the MEMTYPE= option, only specified types are listed. If you specify the DETAILS option, PROC DATASETS prints these additional columns of information: **`Obs, Entriesor Indexes`**, **`Vars`**, and **`Label`**.

Directory Listing as Output

The CONTENTS statement lists the directory of the procedure input library if you use the DIRECTORY option or specify DATA=_ALL_.

If you want only a directory, use the NODS option and the _ALL_ keyword in the DATA= option. The NODS option suppresses the description of the tables; only the directory appears in the output.

Note: The CONTENTS statement does not put a directory in an output table. If you try to create an output table using the NODS option, you receive an empty output table. Use the SQL procedure to create a table that contains information about a library.

Note: If you specify the ODS RTF destination, the PROC DATASETS output goes to both the log and the ODS output area. The NOLIST option suppresses output to both. To see the output only in the log, use the ODS EXCLUDE statement by specifying the member directory as the exclusion.

Procedure Output

The CONTENTS Statement

The only statement in PROC DATASETS that produces procedure output is the CONTENTS statement. This section shows the output from the CONTENTS statement for the Health library and the Group table, which is shown in the following output.

Only the items in the output that require explanation are discussed.

Table Attributes

Here are descriptions of selected fields shown in the following output:

`Member Type`
is the type of library member (DATA).

`Variables`
is the number of columns in the table.

`Created`
indicates the date and time that the table was created.

`Last Modified`
indicates the date and time that the table was last modified.

`Scope`
The CAS engine allows a data set to be local (session) or promoted (global) or both. The DATASETS procedure output displays a Scope column for the CAS engine. The column has a value of Session or Global. If both values are present in the CAS engine, then only the local (session) data set can be accessed.

`Protection`
blank

`Data Set Type`
blank

`Observations`
is the total number of rows currently in the table. If the number of rows exceeds the largest integer value that can be represented in a double precision floating point number, the count is shown as missing.

`Deleted Observations`
0

`Compressed`
indicates whether the table is compressed.

`Sorted`
no

`Data Representation`
is the format in which data is represented on a computer architecture or in an operating environment.

`Encoding`
is the encoding value. Encoding is a set of characters (letters, logograms, digits, punctuation, symbols, control characters, and so on). The set of characters have been mapped to numeric values (called code points) that can be used by computers. The code points are assigned to the characters in the character set when you apply an encoding method.

Output 4.2 *Attributes of the Cars Table*

Using the DIRECTORY Option

The CONTENTS Procedure

Data Set Name	MYCAS.CARS	**Observations**	428
Member Type	DATA	**Variables**	15
Engine	CAS	**Indexes**	0
Created	08/11/2016 10:56:31	**Observation Length**	160
Last Modified	08/11/2016 10:56:31	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

Alphabetic List of Variables and Attributes

Here are descriptions of selected columns in the previous output:

#

is the logical position of each variable in the row. This number is assigned to the variable when the variable is defined.

`Variable`

is the name of each variable. By default, variables appear alphabetically with uppercase and lowercase names listed separately.

Note: Variable names are sorted such that X1, X2, and X10 appear in that order and not in the true collating sequence of X1, X10, and X2. Variable names that contain an underscore and digits might appear in a nonstandard sort order. For example, P25 and P75 appear before P2_5.

Type
specifies the type of variable.

Len
specifies the variable's length, which is the number of bytes used to store each of a variable's values in a table.

Format
specifies the format.

Label
specifies the label.

PROC DATASETS and the Output Delivery System (ODS)

Most procedures send their messages to the log and their procedure results to the output. PROC DATASETS is unique because it sends procedure results to both the log and the procedure output table. When the interface to ODS was created, it was decided that all procedure results (from both the log and the procedure output table) should be available to ODS. In order to implement this feature and maintain compatibility with earlier releases, the interface to ODS had to be slightly different from the usual interface.

By default, the PROC DATASETS statement itself produces two output objects: Members and Directory. These objects are routed to the log. The CONTENTS statement produces three output objects by default: Attributes, EngineHost, and Variables. (The use of various options adds other output objects.) These objects are routed to the procedure output table. If you open an ODS destination (such as HTML, RTF, or PRINTER), all of these objects are, by default, routed to that destination.

You can use ODS SELECT and ODS EXCLUDE statements to control which objects go to which destination, just as you can for any other procedure.

ODS Table Names

PROC DATASETS and PROC CONTENTS assign a name to each table that they create. You can use these names to reference the table when using the Output Delivery System (ODS) to select tables and create output tables.

PROC CONTENTS generates the same ODS tables as PROC DATASETS with the CONTENTS statement.

Table 4.2 *ODS Tables Produced by the DATASETS Procedure without the CONTENTS Statement*

ODS Table	Description	Generates Table
Directory	General library information	Unless you specify the NOLIST option
Members	Library member information	Unless you specify the NOLIST option

Table 4.3 *ODS Table Names Produced by PROC CONTENTS and PROC DATASETS with the CONTENTS Statement*

ODS Table	Description	Generates Table
Attributes	Data set attributes	Unless you specify the SHORT option
Directory	General library information	If you specify DATA=<*libref*>_ALL_ or the DIRECTORY option
Members	Library member information	If you specify DATA=<*libref*>_ALL_ or the DIRECTORY option
Position	A detailed listing of variables by logical position in the table	If you specify the VARNUM option and you do not specify the SHORT option
PositionShort	A concise listing of variables by logical position in the table	If you specify the VARNUM option and the SHORT option
Variables	A detailed listing of variables in alphabetical order	Unless you specify the SHORT option
VariablesShort	A concise listing of variables in alphabetical order	If you specify the SHORT option

Output Tables

The CONTENTS Statement

The CONTENTS statement is the only statement in the DATASETS procedure that generates output tables.

The OUT= Table

The OUT= option in the CONTENTS statement creates an output table. Each variable in each DATA= table has one row in the OUT= table. The fields are blank (if character values) and are missing if not character values. Here are the variables in the output table:

CHARSET
: the character set used to sort the table. The value is ANSI. A blank appears if the table does not have a sort indicator stored with it.

COLLATE
: the collating sequence used to sort the table. A blank appears if the sort indicator for the input table does not include a collating sequence.

COMPRESS
: indicates whether the table is compressed.

CRDATE
: date the table was created.

DELOBS
: number of rows marked for deletion in the table.

ENCRYPT
indicates whether the table is encrypted.

ENGINE
name of the method used to read from and write to the table.

FLAGS
indicates whether the variables in an SQL view are protected (**P**) or contribute (**C**) to a derived variable.

P
indicates the variable is protected. The value of the variable can be displayed but not updated.

C
indicates whether the variable contributes to a derived variable.

The value of FLAG is blank if **P** or **C** does not apply to an SQL view or if it is a table view.

FORMAT
variable format. The value of FORMAT is a blank if you do not associate a format with the variable.

FORMATD
number of decimals that you specify when you associate the format with the variable. The value of FORMATD is 0 if you do not specify decimals in the format.

FORMATL
format length. If you specify a length for the format when you associate the format with a variable, the length that you specify is the value of FORMATL. You do not specify a length for the format when you associate the format with a variable,. The value of FORMATL is the default length of the format if you use the FMTLEN option and 0 if you do not use the FMTLEN option.

GENMAX
maximum number of versions for the generation group.

GENNEXT
the next generation number for a generation group.

GENNUM
the version number.

IDXCOUNT
number of indexes for the table.

IDXUSAGE
use of the variable in indexes. Possible values are

NONE
the variable is not part of an index.

SIMPLE
the variable has a simple index. No other variables are included in the index.

COMPOSITE
the variable is part of a composite index.

BOTH
the variable has a simple index and is part of a composite index.

INFORMAT
variable informat. The value is a blank if you do not associate an informat with the variable.

INFORMD
number of decimals that you specify when you associate the informat with the variable. The value is 0 if you do not specify decimals when you associate the informat with the variable.

INFORML
informat length. If you specify a length for the informat when you associate the informat with a variable, the length that you specify is the value of INFORML. You do not specify a length for the informat when you associate the informat with a variable. The value of INFORML is the default length of the informat if you use the FMTLEN option and 0 if you do not use the FMTLEN option.

JUST
justification (0=left, 1=right).

LABEL
variable label (blank if none given).

LENGTH
variable length.

LIBNAME
libref used for the data library.

MEMLABEL
label for this SAS table (blank if no label).

MEMNAME
SAS table that contains the variable.

MEMTYPE
library member type.

MODATE
date the table was last modified.

NAME
variable name.

NOBS
number of rows in the table.

NODUPKEY
indicates whether the NODUPKEY option was used in a PROC SORT statement to sort the input table.

NPOS
physical position of the first character of the variable in the table.

POINTOBS
indicates whether the table can be addressed by row.

PROTECT
the first letter of the level of protection. The value for PROTECT is one or more of the following:

A
indicates the table is alter-protected.

R
indicates the table is read-protected.

`W`
indicates the table is write-protected.

REUSE
indicates whether the space made available when rows are deleted from a compressed table should be reused. If the table is not compressed, the REUSE variable has a value of NO.

SORTED
the value depends on the sorting characteristics of the input table. Here are the possible values:

. (period)
for not sorted.

0
for sorted but not validated.

1
for sorted and validated.

SORTEDBY
the value depends on that variable's role in the sort. Here are the possible values:

. (period)
if the variable was not used to sort the input table.

n
where *n* is an integer that denotes the position of that variable in the sort. A negative value of *n* indicates that the table is sorted by the descending order of that variable.

TRANSCODE
indicates whether the variable is transcoded.

TYPE
type of the variable (1=numeric, 2=character).

TYPEMEM
special table type (blank if no TYPE= value is specified).

VARNUM
variable number in the table. Variables are numbered in the order in which they appear.

Chapter 5

DELETE Procedure

Overview: DELETE Procedure

The DELETE procedure deletes CAS tables. Use PROC DELETE to do the following:

- delete either permanent or temporary tables
- delete a list of CAS tables with the same name and a numeric suffix, such as

```
proc delete data=x1-x3;
run;
```

Concepts: DELETE Procedure

One of the benefits of using the DELETE procedure is that it does not use the in-memory directory to delete CAS tables. As a result, the DELETE procedure is faster. The DELETE procedure produces no printed output.

Syntax: DELETE Procedure

Note: MEMTYPE= does not apply to CAS engine tables.

PROC DELETE <LIBRARY=*libref*>DATA=*table(s)* <*option(s)*>;

Statement	Task	Example
PROC DELETE	Deletes CAS tables from a CAS engine library	Ex. 2, Ex. 2

PROC DELETE Statement

Deletes CAS tables from a CAS engine library.

Syntax

PROC DELETE LIBRARY=*libref*
DATA=*table-name(s)*;

Summary of Optional Arguments

LIBRARY=*libref*
specifies a name that is associated with the location of a CAS engine library.

Required Argument

DATA= ***table-name(s)***
specifies one or more CAS tables that you want to delete.

Note: You can also use a numbered range list. You cannot use a colon list.

Optional Argument

LIBRARY=*libref*
specifies a name that is associated with the location of a CAS engine library.

Alias LIB=

Details

The Basics

If you attempt to delete a CAS table that does not exist in the procedure input CAS engine library, a message is written to the log and processing continues. If NOWARN is used, no message is issued.

Examples: DELETE Procedure

Example 1: Deleting Several CAS Tables

Features: PROC DELETE statement options
DATA=

Details

This example demonstrates how to delete CAS tables.

Program

```
proc delete data=mycas.A mycas.B mycas.C;
run;
```

Program Description

Delete CAS tables named A, B, and C from a caslib named Mycas.

```
proc delete data=mycas.A mycas.B mycas.C;
run;
```

Example 2: Using the LIBRARY= Option

Features: PROC DELETE statement options
DATA=
LIB=

Details

The following statement deletes a CAS table from a specific caslib.

Program

```
proc delete lib=mycas data=A;
run;
```

Program Description

Deletes the A table that is in the specified caslib named Mycas. The alias for the LIBRARY= option is LIB=.

```
proc delete lib=mycas data=A;
run;
```

Chapter 6

DS2 Procedure

Overview: DS2 Procedure

The DS2 procedure enables you to submit DS2 language statements from a Base SAS session or a SAS Cloud Analytic Services session. The procedure enables the requests to be processed by the DS2 data access technology that supports a scalable, threaded, high-performance, and standards-based way to access, manage, and share relational data.

With the procedure, you can develop the language code in an isolated development or test environment before use in a production, multi-user server context. In addition, the procedure enables the DS2 language to be used in large, complex applications that submit jobs to be executed using SAS data access services.

DS2 is a SAS programming language that is appropriate for advanced data manipulation. DS2 is included with Base SAS and shares core features with the SAS DATA step. DS2 exceeds the DATA step by adding variable scoping, user-defined methods, ANSI SQL

data types, and user-defined packages. The DS2 SET statement accepts embedded FedSQL syntax, and the runtime-generated queries can exchange data interactively between DS2 and any supported database. This allows SQL preprocessing of input tables, which effectively combines the power of the two languages.

For information about the DS2 language, see *SAS Viya: DS2 Language Reference*.

Concepts: DS2 Procedure

Benefits of the DS2 Language

DS2 programs are written for applications that

- benefit from using the new expressions or write methods or packages available in the DS2 syntax
- need to execute the SAS FedSQL language from within the DS2 program
- require the precision that results from using ANSI SQL data types
- execute outside of a SAS session (for example, on SAS High-Performance Analytics Server, SAS Federation Server, or SAS Cloud Analytic Services)
- take advantage of threaded processing in products such as SAS High-Performance Analytics Server, SAS Enterprise Miner, and SAS Cloud Analytic Services

Data Source Support

In SAS Viya, PROC DS2 can access the following data sources:

- SAS data sets
- SAS Cloud Analytic Services tables

Syntax: DS2 Procedure

PROC DS2 *<option(s)>*;
  …DS2 language statements
  RUN;
  RUN CANCEL;
  QUIT;

Statement	Task	Example
PROC DS2	Specify that the subsequent input is DS2 language statements	Ex. 1, Ex. 2, Ex. 4
RUN CANCEL	Cancel the previous DS2 language statements	Ex. 3

PROC DS2 Statement

Specifies that the subsequent input is DS2 language statements.

Interactions: The DS2 procedure requires the RUN statement to submit DS2 statements. That is, SAS reads the program statements that are associated with one task until it reaches a RUN statement.

By default, the procedure processes nonexistent numeric values as SAS missing values.

By default, you cannot overwrite an existing table. To specify that an output table should be deleted before a replacement output table is created, use the OVERWRITE=YES table option. For information about the OVERWRITE= table option, see *SAS Viya: DS2 Language Reference*.

Examples: "Example 1: Creating a SAS Data Set" on page 87
"Example 2: Run a DS2 Program in CAS" on page 89
"Example 4: Creating Tables Based on a Condition" on page 93

Syntax

PROC DS2 <*SAS-connection-option* | *CAS-connection-option*><*processing-option(s)*>;

Summary of Optional Arguments

CAS Connection

SESSREF=*session-name*
: specifies to run the DS2 statements in a SAS Cloud Analytic Services (CAS) session. The CAS session is identified by its session name.

SESSUUID="*session-uuid*"
: specifies to run the DS2 statements in a SAS Cloud Analytic Services (CAS) session. The CAS session is identified by its universally unique identifier (UUID).

General Processing

ERRORSTOP | NOERRORSTOP
: specifies whether the procedure stops executing if it encounters an error.

LABEL | NOLABEL
: specifies whether to use the column label or the column name as the column heading.

MEMSIZE=*n* | *n*M | *n*G
: specifies a limit for the amount of memory that is used for an underlying query (such as a SELECT statement), so that allocated memory is available to support other PROC DS2 operations.

NUMBER
: specifies to include a column named Row, which is the row (observation) number of the data as the rows are retrieved.

SCOND=WARNING | NONE | NOTE | ERROR
: specifies the level of messages that PROC DS2 displays in the SAS log for the DS2 variable declaration strict mode, which requires that every variable must be declared in the DS2 program.

STIMER
: specifies to write a subset of system performance statistics, such as time-elapsed statistics, to the SAS log.

XCODE=ERROR | WARNING | IGNORE
: controls the behavior of the SAS session when an NLS transcoding failure occurs.

SAS Connection

LIBS=*libref* | (*libref1 libref2 ...librefn*)
: restricts the data source connection to the specified libref(s).

NOLIBS CONN= "*connection-string*"
: overrides the default data source connection with a specified connection string.

Optional Arguments

ERRORSTOP | NOERRORSTOP

specifies whether the procedure stops executing if it encounters an error. In a batch or noninteractive session, ERRORSTOP instructs the procedure to stop executing the statements but to continue checking the syntax after it has encountered an error. NOERRORSTOP instructs the procedure to execute the statements and to continue checking the syntax after an error occurs.

Default NOERRORSTOP in an interactive SAS session; ERRORSTOP in a batch or noninteractive session

Tips ERRORSTOP has an effect only when SAS is running in the batch or noninteractive execution mode.

NOERRORSTOP is useful if you want a batch job to continue executing SQL procedure statements after an error is encountered.

LABEL | NOLABEL

specifies whether to use the column label or the column name as the column heading.

Default LABEL

Interactions If a column does not have a label, the procedure uses the column's name as the column heading.

A column alias overwrites the label or column name as the column heading.

LIBS=*libref* | (*libref1 libref2 ...librefn*)

restricts the data source connection to the specified libref(s). By default, PROC DS2 builds a data source connection that includes all currently assigned librefs to pass to the DS2 program. When the LIBS= option is specified, only the specified librefs are included in the data source connection; all other librefs are ignored.

When you specify a list of librefs, the order of the list defines the library order. The Work library is implicitly included as the first library in all lists and is the default library for data references that do not specify a libref.

Alias LIBNAMES=

Restriction LIBS= cannot be used with SESSREF= (or SESSUID=).

Interactions The LIBS= option accomplishes the same result as the NOLIBS CONN= option. The use of LIBS= is recommended over NOLIBS CONN= because you do not need to know how to write a data source connection string to use LIBS=.

If both LIBS= and NOLIBS CONN= are specified, the last option in the procedure statement is applied.

Tip If you are curious about how LIBS= affects library assignments, set the MSGLEVEL=i system option before running a PROC DS2 request with LIBS=. The option will produce Include and Ignore messages for each of the LIBNAME statements that are processed in the procedure request.

Example The following PROC DS2 procedure statement specifies to create a data source connection that uses only librefs MyLib3 and MyLib4:

```
proc ds2 libs=(mylib3 mylib4);
```

MEMSIZE=*n* | *n*M | *n*G

specifies a limit for the amount of memory that is used for an underlying query (such as a SELECT statement), so that allocated memory is available to support other PROC DS2 operations. Specify the memory limit in multiples of 1 (bytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). For example, the value **23m** specifies 24,117,248 bytes of memory.

Alias STMTMEMLIMIT=

Tip Generally, specifying a memory limit is not necessary unless DS2 reports a memory problem error.

NOLIBS CONN= "*connection-string*"

overrides the default data source connection with a specified connection string. By default, PROC DS2 builds a data source connection that includes all currently assigned librefs to pass to the DS2 program. NOLIBS turns off the default data source connection. CONN= specifies an alternate data source connection string.

Connection-string can have the following attributes.

CATALOG="*catalog-identifier*";

specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. A catalog name can be up to 32 characters long.

DRIVER="*driver-name*";

specifies the data source that you want to connect to.

SCHEMA=*value*

specifies a SCHEMA= in which to create or read data.

Value must be in the following form:

```
(NAME=value; PRIMARYPATH=value)
```

NAME=*identifier*

specifies an arbitrary identifier for an SQL schema. Any identifier is valid (for example, name=myfiles). The schema identifier is an alias for the physical location of the SAS library, which is much like the Base SAS libref. A schema name must be a valid SAS name and can be up to 32 characters long. You must specify a schema identifier.

PRIMARYPATH={*pathname*}
specifies the physical location for the SAS library. In most operating environments, this is a directory path. You must specify a primary path.

Restriction NOLIBS CONN= cannot be used with SESSREF= (or SESSUID=).

Interactions NOLIBS CONN= accomplishes the same result as LIBS= on page 74. Use of LIBS= is recommended over NOLIBS CONN= because you do not need to know how to write a data source connection string to use LIBS=.

If both NOLIBS CONN= and LIBS= are specified, the last option in the procedure statement is applied.

Tip Data source connection strings are difficult to write. If you set the MSGLEVEL=i system option and rerun a failed request, the data source connection strings that were generated by the procedure are written to the SAS log. Use the generated connection strings as a guideline.

Examples Here is a PROC DS2 procedure statement that uses NOLIBS CONN= to specify a connection string for Base SAS data:

```
proc ds2 nolibs
   conn="driver=base;
         catalog=base;
         schema=(name=base;primarypath=.)";
```

Here is a procedure statement that specifies two paths:

```
proc ds2 nolibs
 conn="(driver=base;
catalog=work;
schema=(name=work;primarypath={%sysfunc(pathname(work))}));
(driver=base;
catalog=base;
schema=(name=base;primarypath='c:\temp\base'))";
```

NUMBER
specifies to include a column named Row, which is the row (observation) number of the data as the rows are retrieved.

Default No row numbers.

Restriction This option is not supported with SESSREF= (or SESSUUID=).

SCOND=WARNING | NONE | NOTE | ERROR
specifies the level of messages that PROC DS2 displays in the SAS log for the DS2 variable declaration strict mode, which requires that every variable must be declared in the DS2 program. For more information about the DS2 variable declaration strict mode, see the *SAS Viya: DS2 Language Reference*.

WARNING
writes warning messages to the SAS log.

NONE
no messages are written to the SAS log.

NOTE
writes notes to the SAS log.

ERROR
writes error messages to the SAS log.

Default The default is determined by the DS2SCOND= system option. The default for DS2SCOND= is WARNING. For information about the DS2SCOND= system option, see *SAS Viya: DS2 Language Reference*.

Interaction Specifying the SCOND= option in the PROC DS2 statement takes precedence over the DS2SCOND= system option.

SESSREF=*session-name*
specifies to run the DS2 statements in a SAS Cloud Analytic Services (CAS) session. The CAS session is identified by its session name. By default, PROC DS2 builds a data source connection that includes all currently assigned librefs to pass to the DS2 program. The SESSREF= option overrides the default behavior and runs the request in the specified CAS session instead. The CAS session must have been previously established by using the CAS statement. If the specified CAS session does not exist, the procedure terminates. You can only connect to one CAS session at a time.

Interactions Use SESSREF= or SESSUUID= to connect to SAS Cloud Analytic Services. If both options are specified, the last option in the procedure statement is applied.

If both SESSREF= and LIBS= (or NOLIBS CONN=) are specified in the procedure statement, SESSREF= is applied and the other option is ignored.

Example Here is a PROC DS2 procedure statement that specifies SESSREF=:

```
proc ds2 sessref=mysess;
```

SESSUUID="*session-uuid*"
specifies to run the DS2 statements in a SAS Cloud Analytic Services (CAS) session. The CAS session is identified by its universally unique identifier (UUID). By default, PROC DS2 builds a data source connection that includes all currently assigned librefs to pass to the DS2 program. The SESSUUID= option overrides the default behavior and runs the request in the specified CAS session instead. The CAS session must have been previously established by using the CAS statement. The CAS statement generates a UUID value. If the specified CAS session does not exist, the procedure terminates. You can only connect to one CAS session at a time.

Interactions Use SESSREF= or SESSUUID= to connect to SAS Cloud Analytic Services. If both options are specified, the last option in the procedure statement is applied.

If both SESSUUID= and LIBS= (or NOLIBS CONN=) are specified in the procedure statement, SESSUUID= is applied and the other option is ignored.

Example Here is a PROC DS2 procedure statement that specifies SESSUUID=:

```
proc ds2 sessuuid="76904741-fb09-554d-a8de-6cbce2a0e0e5";
```

The UUID value can be enclosed in single or double quotation marks.

STIMER
specifies to write a subset of system performance statistics, such as time-elapsed statistics, to the SAS log. When STIMER is in effect, the procedure writes to the SAS log a list of computer resources used for each step and the entire SAS session.

Default No performance statistics are written to the SAS log.

Interaction If the SAS system option FULLSTIMER is in effect, the complete list of computer resources is written to the SAS log.

XCODE=ERROR | WARNING | IGNORE
controls the behavior of the SAS session when an NLS transcoding failure occurs. Transcoding failures can occur during row input or output operations, or during string assignment. Transcoding is the process of converting character data from one encoding to another encoding.

ERROR
specifies that a run-time error occurs, which causes row processing to halt. An error message is written to the SAS log. This is the default behavior.

WARNING
specifies that the incompatible character is set to a substitution character. A warning message is written to the SAS log.

IGNORE
specifies that the incompatible character is set to a substitution character. No messages are written to the SAS log.

Default ERROR

RUN CANCEL Statement

Cancels the previous DS2 language statements.

Tip: The RUN CANCEL statement is useful if you enter a typographical error.

Example: "Example 3: Terminating the Current Step in Line Prompt Mode" on page 92

Syntax

RUN CANCEL;

Using the DS2 Procedure

Data Source Connection

PROC DS2 can execute requests in a SAS session, or it can execute requests in a SAS Cloud Analytic Services session. By default, PROC DS2 connects to a data source by using the SAS data access services. You can override this default behavior by specifying connection options. The DS2 procedure is not affected by the CASNAME= system option.

Understanding the Default Data Source Connection

PROC DS2 connects to a data source by using the attributes of currently assigned librefs. Attributes include the physical location of the data, and for some data sources, access

information such as network information used to access the data server, and user identification and password.

You first submit the LIBNAME statement for a SAS engine and then submit PROC DS2. The supported engine includes SAS V9 BASE.

This example illustrates how PROC DS2 accesses a data source by using the attributes of a previously assigned libref. The LIBNAME statement assigns the libref MyFiles, specifies the BASE engine, and then specifies the physical location for the SAS data set. The DS2 program then creates the SAS data set MyFiles.Table1 at the location specified in the LIBNAME statement.

```
libname myfiles base 'C:\myfiles\base';

proc ds2;
   data myfiles.table1;
      dcl double j j2;
      method run();
         do j = 1 to 1000;
            j2 = 2*j;
            output;
         end;
      end;
   enddata;
run;
quit;
```

The procedure builds a data source connection string that includes all active librefs and sends it to the DS2 program. You reference a particular library by specifying its libref in a two-part table name in the form *libref.table-name*. If you do not specify a libref, the table is created in the SAS Work library.

PROC DS2 uses libref attributes for connection information only (such as physical location). PROC DS2 does not use libref attributes that define behavior. For example, if a previously submitted LIBNAME statement for the BASE engine specifies that SAS data sets are to be compressed, the compression attribute is not used by the procedure.

Note: PROC DS2 connects immediately, so an error is generated if the LIBNAME statement includes the DEFER=YES option.

Note: To access data created with the SAS 9.4 BASE engine, you must have SAS Viya Bridge software.

Connecting with LIBS=

When multiple librefs are active in the SAS session, you might want to include the LIBS= option in the PROC DS2 statement. LIBS= restricts the data source connection to the specified libref or librefs.

The following example illustrates the use of the LIBS= procedure option. In the example, two librefs are assigned in the SAS session: AllFiles and MyFiles. The LIBS= option specifies to use libref MyFiles only.

```
libname allfiles 'C:\sharedfiles';
libname myfiles base 'C:\myfiles';

proc ds2 libs=myfiles;
   data myfiles.table1;
      dcl double j j2;
```

```
      method run();
         do j = 1 to 1000;
            j2 = 2*j;
            output;
         end;
      end;
   enddata;
run;
quit;
```

When LIBS= is specified, the data source connection string that is generated by SAS data access services includes only information from the specified libref. For more information, see "LIBS=*libref* | (*libref1 libref2 ...librefn*)" on page 74.

NOLIBS CONN= accomplishes the same thing as LIBS=, except you must specify a data source connection string, which is more difficult to do.

Connecting with SESSREF= or SESSUUID=

You connect to a SAS Cloud Analytic Services session by specifying the SESSREF= (or SESSUUID=) procedure option. When either of these options is specified, both the default connection mechanism and the LIBS= and NOLIBS CONN= options are ignored. Instead, the procedure connects to the specified SAS Cloud Analytic Services session.

You must first submit the CAS statement to establish a CAS session. To interact with data in a CAS session, you need a CAS library (caslib). Caslibs behave differently from librefs:

- Caslibs can be personal, pre-defined by an administrator, or added manually. Your level of authorization determines your interaction with each type of caslib.
- When you start a CAS session, the personal caslib CASUSER is automatically allocated and becomes the active caslib for your session, unless other caslibs are defined.
- You can determine what caslibs are available to you by using the CASLIB statement with the LIST argument. If you are authorized, you can add caslibs with the CASLIB statement.
- Pre-defined caslibs often provide pre-loaded data. For the other caslibs, you must create or load the data in CAS before you can access the data using SAS Cloud Analytic Services. You can create data in CAS by using PROC DS2 and the DATA step. You can load existing data into CAS by using the CASUTIL procedure. The PROC CASUTIL LOAD statement supports loading of data from SAS, ODBC, Hadoop, Impala, Teradata, Oracle and PostGres into CAS.

After ensuring that the caslib that you want to access is the active caslib in your CAS session (and loading data if necessary), submit PROC DS2. In the PROC DS2 statement, specify the session name that you defined in the CAS statement in the SESSREF= option. Or specify the session's UUID in the SESSUUID= option.

The following example illustrates the use of the SESSREF= option.

```
cas mysess; 1

proc ds2 sessref=mysess; 2
   data table1;
      dcl double j j2;
      method run();
         do j = 1 to 1000;
```

```
            j2 = 2*j;
            output;
         end;
      end;
   enddata;
run;
quit;
```

1. The CAS statement establishes a CAS session named “MySess”. The CAS statement does not specify a CAS library; therefore, the personal caslib, CASUSER, is used for the operation.
2. PROC DS2 is executed with the SESSREF= procedure option. SESSREF= specifies to connect to SAS Cloud Analytic Services, instead of using the default SAS connection. The procedure connects to the CAS session using the active caslib, CASUSER, and creates a table named Table1. To access existing data with PROC DS2, you must first load the data into the SAS Cloud Analytic Services session.

When only one caslib is active in the CAS session, there is no need to reference the caslib. Use a two-part name when more than one caslib is active.

The CAS tables that you create with PROC DS2 are in-memory tables. That is, the tables are available for the duration of the CAS session and are accessible only to the current session. PROC DS2 does not provide a way to persist a table or to share the table with other CAS sessions. To persist or share a CAS table, use the CASUTIL procedure.

Note: Although CAS tables are in-memory tables, you must specify the OVERWRITE= table option if you want an initial output table in the CAS session to be overwritten by a replacement output table.

For more information, see:

- *SAS Cloud Analytic Services: Fundamentals*
- CAS statement, CASLIB statement, and CASUTIL procedure in *SAS Cloud Analytic Services: Language Reference*
- SESSREF=“SESSREF=*session-name*” on page 77.
- “SESSUUID="*session-uuid*"” on page 77.

Using PROC DS2 in SAS Cloud Analytic Services

SAS Cloud Analytic Services is an alternative environment for processing DS2 requests. When you specify the SESSREF= (or SESSUUID=) option, PROC DS2 prepares and executes your DS2 programs on a SAS Cloud Analytic Services (CAS) server. When you run PROC DS2 with SESSREF= or SESSUUID=, you are actually using the runDS2 action. Unless you are using Lua or Python, it is recommended that you use PROC DS2 to run your DS2 program instead of the runDS2 action. There are advantages of using PROC DS2. For more information, see the runDS2 action in the *SAS Visual Data Mining and Machine Learning System Programming Guide*.

The CAS server is a Symmetric Multi-Processing (SMP) server. A DS2 program executing on the CAS server can perform manipulations on multiple data observations concurrently reducing the time required to process large data sets. Based on the structure of the DS2 program, the DS2 compiler determines which operations can be performed on multiple observations concurrently and which operations must be applied to each observation sequentially.

A DS2 program is classified as either a serial program, a parallel program, or a parallel-serial program. In order to benefit from the CAS server, the DS2 program must be structured as either a DS2 parallel program or a DS2 parallel-serial program.

- A DS2 parallel program contains no operations with data dependencies across observations; therefore, multiple data observations can be processed in parallel. Each CAS worker can process a subset of the input data and generate a subset of the result set.
- A DS2 parallel-serial program contains some operations with data dependencies across observations and some operations without data dependencies. The processing of the operations are divided into two stages: a parallel stage and a serial stage. During the parallel stage, each CAS worker processes a subset of the input data set and generates a subset of an intermediate data set. During the serial stage, one CAS worker processes the complete intermediate data set and generates the complete result set.

Most of the functionality of the DS2 language is supported for use in SAS Cloud Analytic Services. There are some exceptions. For information about the DS2 functionality that is supported in CAS, see *SAS Viya: DS2 Programmer's Guide*.

For an example of how a DS2 parallel program is submitted to CAS, see "Example 2: Run a DS2 Program in CAS" on page 89.

RUN-Group Processing

PROC DS2 supports RUN-group processing. RUN-group processing enables you to submit RUN groups without ending the procedure.

To use RUN-group processing, you start the procedure and then submit multiple RUN-groups. A RUN-group is a group of statements that contains at least one action statement and ends with a RUN statement. As long as you do not terminate the procedure, it remains active and you do not need to resubmit the PROC statement.

Note: When using PROC DS2, DS2 programs are delimited by RUN statements. If additional DS2 code is found after a RUN statement, then this code composes a new, distinct DS2 program from the DS2 program before the previous RUN statement.

To end RUN-group processing, submit a RUN CANCEL statement. Statements that have not been submitted are terminated. To stop the procedure, submit a QUIT statement. Statements that have not been submitted are terminated as well.

Applying DS2 Table Options

When you access a data source with PROC DS2, you can apply DS2 table options in the subsequent DS2 statements. A table option specifies actions that enable you to perform operations on a table such as assigning buffer page size or specifying passwords. A DS2 table option performs much of the same functionality as a Base SAS data set option.

DS2 table options are used to apply options when you access a data source within PROC DS2. For example, the following code applies a table option to the SAS data set to specify the size of a permanent buffer page for the new table:

```
libname myfiles base 'C:\myfiles\Base';

proc ds2;
   data myfiles.table1 (bufsize=16k);
      dcl double j j2;
```

```
      method run();
         do j = 1 to 1000;
            j2 = 2*j;
            output;
         end;
      end;
enddata;
run;
quit;
```

For a list of available table options, see *SAS Viya: DS2 Language Reference*.

Using Macro Variables in a Literal String

Note: The information in this section pertains to using PROC DS2 on Base SAS data sets only. CAS does not support macro variables.

Macro variables enable you to dynamically modify text in a program through symbolic substitution. When you reference a macro variable in a program, the macro processor replaces the reference with the value of the specified macro variable.

With PROC DS2, you can use a macro variable on a subsequent DS2 statement. However, if a macro variable occurs within a literal string, you cannot enclose the string in double quotation marks, which is required by the macro processor to resolve the macro variable reference. You cannot enclose the string in double quotation marks because DS2 statements consider a string enclosed in double quotation marks to be a delimited (case sensitive) identifier such as a table or column name.

To reference a macro variable in a literal string, use the SAS macro function %TSLIT, which overrides the need for double quotation marks around the literal string and puts single quotation marks around the input value. For example, the following statement includes the %TSLIT function to specify the &SYSHOSTNAME macro variable, which returns the host name of the computer on which it is executed:

```
if hostname = %tslit(&syshostname) then ...
```

The %TSLIT macro function is stored in the default autocall macro library. For more information, see “Referencing a Macro Variable in a Delimited Identifier” in the *SAS Viya: DS2 Language Reference*.

DS2 Automatic Variables

The DS2 language sets up automatic variables with certain values after it executes each statement. These automatic variables are useful for subsetting a problem across DS2 threads. They are also useful for providing context when debugging with PUT statements.

Table 6.1 *DS2 Automatic Variables*

Variable	Description
HOSTNAME	Name of the machine that this data program or thread program is running on.
THREADID	Zero-based thread number for this DS2 thread in the running program.

Variable	Description
NTHREADS	Total number of DS2 threads in the running program.

Here is an example of how the automatic variables might be used:

```
proc ds2;
thread thd / overwrite = yes;
  dcl double x;
  method init();
    put 'THREAD: thread' _threadid_ 'on' _hostname_;
  end;
endthread;

data _null_;
  dcl thread thd t;
  method init();
    put 'DATA: thread' _threadid_ 'on' _hostname_;
  end;
  method run();
    set from t threads=8;
  end;
enddata;
run;
quit;
```

Security

SAS Viya supports a security scheme that is implemented by administrators. For more information, see *SAS Cloud Analytic Services: Authorization*.

In addition, PROC DS2 supports password protection for SAS data sets.

Base SAS software enables you to restrict access to SAS data sets by assigning SAS passwords to the files. You can specify three levels of protection: read, write, and alter.

With PROC DS2, you assign or specify a password for a data source using the DS2 table options ALTER=, PW=, READ=, and WRITE=. For example, the following code applies the DS2 table option PW= in order to assign READ, WRITE, and ALTER passwords to a SAS data set:

```
libname myfiles base 'C:\myfiles';

proc ds2;
   data myfiles.table1 (pw=luke);
      dcl double j j2;
      method run();
         do j = 1 to 1000;
            j2 = 2*j;
            output;
         end;
      end;
enddata;
run;
quit;
```

DS2 table options perform much of the same functionality as Base SAS data set options. However, Base SAS data set options are not supported in the PROC DS2 statement. Therefore, DS2 table options must be used to assign or specify passwords when accessing a data source with PROC DS2.

A SAS password does not control access to a SAS file beyond the SAS System. You should use the operating system-supplied utilities and file-system security controls to control access to SAS files outside SAS.

DS2 Data Type Support for SAS Data Sets

In PROC DS2, when you submit DS2 statements, all DS2 language data types are supported. For information about the DS2 data types, see *SAS Viya: DS2 Language Reference*.

However, in a Base SAS session, when you are not submitting PROC DS2, DS2 data types are translated to and from predetermined SAS data types, which are SAS numeric and SAS character. For example, when you submit the CONTENTS procedure on a table that is created with the DS2 language, the DATE data type is reported as a SAS numeric. The following table lists the DS2 data types and how they are translated to and from SAS data types:

Table 6.2 *DS2 Data Type Translation*

DS2 Data Type	SAS Data Type	Description
BIGINT	SAS numeric	Applies the SAS format 20. Because a SAS numeric is a DOUBLE, which is an approximate numeric data type rather than an exact numeric data type, there is potential for loss of precision.
BINARY(*n*)	SAS character	Applies the SAS format $*n*.
CHAR(*n*)	SAS character	Applies the SAS format $*n*.
DATE	SAS numeric	Applies the SAS format DATE9. Valid SAS date values are in the range from 1582-01-01 to 9999-12-31. Dates outside the SAS date range are not supported and are treated as invalid dates.
DECIMAL| NUMERIC(*p*,*s*)	SAS numeric	Applies the SAS format 11.2.
DOUBLE	SAS numeric	
FLOAT	SAS numeric	
INTEGER	SAS numeric	Applies the SAS format 11.
NCHAR(*n*)	SAS character	Applies the SAS format $*n*.
NVARCHAR(*n*)	SAS character	Applies the SAS format $*n*.

DS2 Data Type	SAS Data Type	Description
REAL	SAS numeric	
SMALLINT	SAS numeric	Applies the SAS format 6.
TIME(*p*)	SAS numeric	Applies the SAS format TIME8.
TIMESTAMP(*p*)	SAS numeric	Applies the SAS format DATETIME19.2.
TINYINT	SAS numeric	Applies the SAS format 4.
VARBINARY(*n*)	SAS character	Applies the SAS format $*n*.
VARCHAR(*n*)	SAS character	Applies the SAS format $*n*.

DS2 Data Type Support for CAS Tables

CAS tables support CHAR, DOUBLE, and VARCHAR data types. When writing data to SAS Cloud Analytic Services, some DS2 data types are translated to the CHAR, DOUBLE, and VARCHAR data types, but not all of them. The following table lists the DS2 data types and how they are translated to and from SAS Cloud Analytic Services data types. When you submit the CONTENTS procedure on a CAS table that is created with the DS2 language, the DATE data type is reported as NUM.

Table 6.3 *DS2 Data Type Translation for CAS Tables*

DS2 Data Type	CAS Data Type	Description
BIGINT	DOUBLE	
BINARY(*n*)	Not supported	
CHAR(*n*)	CHAR	Applies the SAS format $*n*. Stores a fixed-length character string, where *n* is the maximum number of characters to store. The maximum number of characters is required to store each value regardless of the actual size of the value. If char(10) is specified and the character string is only five characters long, the value is right-padded with spaces.
DATE	DOUBLE	Applies the SAS format DATE9. Valid SAS date values are in the range from 1582-01-01 to 9999-12-31. Dates outside of the SAS date range are not supported and are treated as invalid dates.
DECIMAL| NUMERIC(*p*,*s*)	Not supported	

DS2 Data Type	CAS Data Type	Description
DOUBLE	DOUBLE	Stores a signed, approximate, double-precision, floating-point number. Allows numbers of large magnitude and permits computations that require many digits of precision to the right of the decimal point. For SAS Cloud Analytic Services, this is a 64-bit double precision, floating-point number.
FLOAT	DOUBLE	
INTEGER	DOUBLE	
NCHAR(*n*)	CHAR	Applies the SAS format $*n*.
NVARCHAR(*n*)	VARCHAR	
REAL	Not supported	
SMALLINT	DOUBLE	
TIME(*p*)	DOUBLE	Applies the SAS format TIME8.
TIMESTAMP(*p*)	DOUBLE	Applies the SAS format DATETIME25.6.
TINYINT	DOUBLE	
VARBINARY(*n*)	Not supported	
VARCHAR(*n*)	VARCHAR	Stores a varying-length character string.

Examples: DS2 Procedure

Example 1: Creating a SAS Data Set

Features: PROC DS2 statement
QUIT statement

Other features: LIBNAME statement
LIBS= procedure option
DS2 language statements
PROC PRINT

Details

This example creates a SAS data set in a Base SAS session by submitting the DS2 procedure, and then submitting DS2 language statements. The output shows the first ten rows of the data set.

Program

```
libname myfiles base 'C:\myfiles';

proc ds2 libs=myfiles;


   data myfiles.basetable;
      declare double j j2;
      method run();
         do j = 1 to 1000;
            j2 = 2*j;
            output;
         end;
      end;
   enddata;

run;

quit;

proc print data=myfiles.basetable (obs=10);
run;
```

Program Description

Assign a library reference to the SAS data set to be created. The following LIBNAME statement assigns the libref MyFiles, specifies the BASE engine, and specifies the physical location for the SAS data set.

```
libname myfiles base 'C:\myfiles';
```

Execute the PROC DS2 statement. The PROC DS2 statement connects to the data source by using the LIBS= option and sets up the environment to submit DS2 language statements.

```
proc ds2 libs=myfiles;
```

Enter the DS2 language statements. The DS2 DATA statement creates an output table named MyFiles.BaseTable. Note that the two-level name in the DATA statement specifies the catalog identifier MyFiles, which is the assigned libref. The DECLARE statement assigns the data type DOUBLE to the variables J and J2. The METHOD statement identifies the RUN system method that is used to create output. The OUTPUT statement writes a row to table MyFiles.BaseTable after each execution of the DO loop.

```
   data myfiles.basetable;
      declare double j j2;
      method run();
         do j = 1 to 1000;
            j2 = 2*j;
            output;
         end;
```

```
        end;
    enddata;
```

Submit the DS2 language statements. The RUN statement submits the DS2 statements. The RUN statement is required. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

```
run;
```

Stop the procedure. The QUIT statement stops the procedure.

```
quit;
```

Print the SAS data set. The PRINT procedure prints the observations in the SAS data set. The OBS= data set option limits the output to 10 observations.

```
proc print data=myfiles.basetable (obs=10);
run;
```

Output: Creating a SAS Data Set

Output 6.1 *PROC PRINT Output of MyFiles.BaseTable*

The SAS System

Obs	j	j2
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10
6	6	12
7	7	14
8	8	16
9	9	18
10	10	20

Example 2: Run a DS2 Program in CAS

Features: CAS system options
CAS statement
CASLIB statement
CASLIB procedure
PROC DS2 statement
SESSREF= procedure option

DS2 parallel program

Details

The following is an example of a DS2 parallel program. A DS2 program with a thread program and a data program that does not contain any data manipulation observations (data program does not contain any statements besides SET FROM and OUTPUT) is a DS2 parallel program. Operations in the thread program are applied to multiple data observations in parallel. Each CAS worker processes a subset of the data set and generates a subset of the result set.

Program

```
options cashost="cloud.example.com" casport=5570;

cas mysess;

caslib casdata datasource=(srctype=path)
  path="testdata/cas";

proc casutil;
  load casdata="cars_single.sashdat" incaslib="casdata" casout="cars_single";
run;

proc ds2 sessref=mysess;

 thread cars_thd / overwrite=yes;
   method run();
     set cars_single;
     if (msrp > 100000) then do;
       put make= model= msrp=;
       output;
     end;
   end;
 endthread;

 data cars_luxury /overwrite=yes;
   dcl thread cars_thd t;
   method run();
     set from t threads=4;
   end;
 enddata;


run;

quit;
```

Program Description

Connect to the CAS server. The CASHOST= and CASPORT= options specify to connect to the CAS server at cloud.example.com using port 5570.

```
options cashost="cloud.example.com" casport=5570;
```

Establish a CAS session. The CAS statement specifies to start a CAS session named MySess.

```
cas mysess;
```

Define a CAS library to access your input data. We want to access a file named Cars_Single.sashdat, which is located in the Testdata/Cas subdirectory of the computer on which the CAS server is running. The CASLIB statement assigns caslib CasData to the directory location. To access a directory on a different computer, specify an absolute pathname.

```
caslib casdata datasource=(srctype=path)
  path="testdata/cas";
```

Load the table into CAS for processing. The CASUTIL procedure is used to load table Cars_Single.sashdat into the CAS session. The CASOUT= parameter assigns the loaded table the name Cars_Single.

```
proc casutil;
  load casdata="cars_single.sashdat" incaslib="casdata" casout="cars_single";
run;
```

Issue the PROC DS2 statement and specify the SESSREF= procedure option. SESSREF= instructs the procedure to process the request using the SAS Cloud Analytic Services session named MySess instead of using SAS data access services.

```
proc ds2 sessref=mysess;
```

Enter the DS2 language statements. The DS2 THREAD statement creates a thread program named Cars_Thd that specifies criteria for selecting data from loaded table Cars_Single. The DS2 DATA statement creates an output table, Cars_Luxury, and specifies to set the results of the thread program as the content of the new table. The data program specifies to use four threads to execute the thread program.

```
thread cars_thd / overwrite=yes;
  method run();
    set cars_single;
    if (msrp > 100000) then do;
      put make= model= msrp=;
      output;
    end;
  end;
endthread;

data cars_luxury /overwrite=yes;
  dcl thread cars_thd t;
  method run();
    set from t threads=4;
  end;
enddata;
```

Submit the DS2 language statements. The RUN statement submits the DS2 statements. The RUN statement is required. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

```
run;
```

Stop the procedure. The QUIT statement stops the procedure.

```
quit;
```

Example 3: Terminating the Current Step in Line Prompt Mode

Features: PROC DS2 statement
RUN CANCEL statement
QUIT statement

Other features: No procedure options
DS2 language statements

Details

The following example shows the usefulness of the RUN CANCEL statement in a line prompt mode session. The sixth statement in the code contains an invalid value for the column (Z instead of Y). RUN CANCEL ends the PROC DS2 step and prevents it from executing.

Because the DS2 procedure statement is submitted without options, the currently assigned librefs from the SAS session are used to process the request.

Program

```
libname myfiles base 'C:\myfiles';

proc ds2;
data myfiles.xy_data;
   declare double x y;
   method init();
      do x = 1 to 5;
         z = 2*x;
      end;
   end;
enddata;
run cancel;
quit;
```

Log 6.1 *SAS Log Showing the Canceled PROC DS2 Step*

```
17   proc ds2;
NOTE: Writing HTML Body file: sashtml1.htm
18   data myfiles.xy_data;
19      dcl double x y;
20      method init();
21         do x = 1 to 5;
22            z = 2*x;
23         end;
24      end;
25   enddata;
26   run cancel;
NOTE: DS2 query cancelled per user request.
27   quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           14.33 seconds
      cpu time            0.71 seconds
```

Example 4: Creating Tables Based on a Condition

Features: PROC DS2 statement
QUIT statement

Other features: DS2 language statements

Details

This example illustrates how to create tables based on a condition. Programs 1 and 2 create two tables, Dept1_Items and Dept2_Items, that hold costs for items used by two departments. The third program creates two tables, Highcosts and Lowcosts, based on the costs of the items in the two tables. Programs 4 and 5 write the contents of the Highcosts and Lowcosts tables. The procedure is executed using the default data access services.

Program

```
proc ds2;
/* Program 1 */
data dept1_items (overwrite=yes);
   dcl varchar(20) item;
   dcl double cost;
   method init();
      item = 'staples';   cost =  1.59; output;
      item = 'pens';      cost =  3.26; output;
      item = 'envelopes'; cost = 11.42; output;
   end;
enddata;
run;
/* Program 2 */
```

```
data dept2_items (overwrite=yes);
   dcl varchar(20) item;
   dcl double cost;
   method init();
      item = 'erasers'; cost =  5.43; output;
      item = 'paper';   cost = 26.92; output;
      item = 'toner';   cost = 62.29; output;
   end;
enddata;
run;
/* Program 3 */
data lowCosts (overwrite=yes) highCosts (overwrite=yes);
   method run();
      set dept1_items dept2_items;
      if cost <= 10.00 then
         output lowCosts;
      else
         output highCosts;
      end;
enddata;
run;
/* Program 4 */
data;
   method run();
      set lowCosts;
   end;
enddata;
run;
/* Program 5 */
data;
   method run();
       set highCosts;
   end;
enddata;
run;
quit;
```

Output: Creating Tables Based on a Condition

Output 6.2 *Output of the Costs Tables*

The SAS System

item	cost
staples	1.59
pens	3.26
erasers	5.43

The SAS System

item	cost
envelopes	11.42
paper	26.92
toner	62.29

Chapter 7

EXPORT Procedure

Overview: Export Procedure

What does the EXPORT Procedure Do?

The EXPORT procedure reads data from a SAS data set and writes it to an external data source. In SAS, external data sources include delimited files and JMP files.

In delimited files, a delimiter can be a blank, comma, or tab that separates columns of data values. If you have a license for SAS/ACCESS Interface to PC Files, you can also export to additional file formats, such as to a Microsoft Access database for DBMS=ACCESS, Microsoft Excel workbook, DBF file, and Lotus spreadsheets. MDB and ACCDB files are not supported. For more information, see *SAS/ACCESS Interface to PC Files for SAS Viya: Reference*.

In SAS, you can export a SAS data set to a JMP 7 or later file, and JMP variables can be up to 255 characters long. Extended attributes are now used automatically, and the META= statement is no longer supported for JMP files. For more information, see "JMP Files" in *SAS/ACCESS Interface to PC Files for SAS Viya: Reference*.

The EXPORT procedure uses one of these methods to export data:

- generated DATA step code

- generated SAS/ACCESS code
- translation engines

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

Format Catalog Encodings in SAS Viya

SAS Viya supports only the UTF-8 encoding.

For more information about the encodings of format catalogs, see *Migrating Data to UTF-8 for SAS Viya* and *SAS Viya FAQ for Processing UTF-8 Data*.

Support for the VARCHAR Data Type

PROC EXPORT supports the VARCHAR data type in CAS. VARCHAR stores a character variable that can have a varying length. The length that you specify for the variable represents the maximum number of characters that you want to store.

The VARCHAR data type is similar to the CHAR data type. CHAR variables have a length that is measured in terms of bytes. VARCHAR variables have a length that is measured in terms of characters rather than bytes. For information about using VARCHAR, see "Data Types Supported in the CAS DATA Step" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

In the following example, the CAS engine is used with the LENGTH statement to create a VARCHAR variable and a CHAR variable. The VARCHAR variable, X, has a length of 30 and the CHAR variable, Y, also has a length of 30.

```
libname mycas cas;
data mycas.string;
   length x varchar(30);
   length y $30;
   x = 'abc'; y = 'def';
run;
proc contents data=mycas.string; run;
```

Here is the output that the code produces.

The CONTENTS Procedure

Data Set Name	MYCAS.STRING	Observations	1
Member Type	DATA	Variables	2
Engine	CAS	Indexes	0
Created	09/11/2016 13:48:30	Observation Length	48
Last Modified	09/11/2016 13:48:30	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	x	Varchar	30
2	y	Char	30

Syntax: EXPORT Procedure

Restrictions: The EXPORT procedure is available for the LINUX environment.

The EXPORT procedure is not supported by the CAS engine. You cannot export a CAS table. For example, the following code generates an error:

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

proc export table="mycas.air" dbms=cas;
run;
```

A pathname for a file can have a maximum length of 201 characters.

Supports: Output can be saved to delimited files and JMP files.

See: For information about using VARCHAR, see "Data Types Supported in the CAS DATA Step" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

PROC EXPORT DATA=<*libref.*>*SAS data set* <(*SAS data set option(s)*)>
    OUTFILE="*filename*" | OUTTABLE="*tablename*"
    <DBMS=*identifier*> <REPLACE> <LABEL>;
statements for exporting to delimited files
    DELIMITER=*char* | '*nn*'x;
    PUTNAMES=YES | NO;
statements for exporting to JMP files
    DBENCODING=*12-char SAS encoding-value*;
    FMTLIB=<*libref.*>*format-catalog*;

Statement	Task	Example
PROC EXPORT	Export SAS data sets to an external data file	Ex. 1, Ex. 2
DBENCODING	Indicate the encoding used to save data in JMP files	
DELIMITER	Specify the delimiter to separate columns of data in the delimited output file	Ex. 1
FMTLIB	Write SAS format values defined in the format catalog to the JMP file for the value labels	
PUTNAMES	Write the SAS variable names as column headings to the first row of the exported data file.	Ex. 3

PROC EXPORT Statement

Exports SAS data sets to an external data file.

Syntax

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

Summary of Optional Arguments

(*SAS data set option(s)*)
specifies SAS data set options.

DBMS=*identifier*
specifies the type of data to export.

LABEL
specifies a variable label name.

REPLACE
overwrites an existing file.

Required Arguments

DATA= <*libref.*> *SAS data set*
identifies the input SAS data set with either a one- or two-level SAS name (library and member name). If you specify a one-level name, by default, the EXPORT procedure uses either the USER library (if assigned) or the WORK library.

The EXPORT procedure can export a SAS data set only if the data target supports the format of a SAS data set. The amount of data must also be within the limitations of the data target. For example, some data files have a maximum number of rows or columns. Some data files cannot support SAS user-defined formats and informats. If the SAS data set that you want to export exceeds the limits of the target file, the EXPORT procedure might not be able to export 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.

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.

Default If you do not specify a SAS data set to export, the EXPORT procedure uses the most recently created SAS data set. SAS keeps track of the data sets with the system variable _LAST_. To be certain that the EXPORT procedure uses the correct data set, you should identify the SAS data set.

Examples "Example 1: Exporting to a Delimited External Data Source" on page 104

"Example 2: Exporting a Subset of Observations to a CSV File" on page 110

OUTFILE="*filename*" | "*fileref*"

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

If you specify a fileref, or if the complete path and filename do not include special characters (such as the backslash in a path), lowercase characters, or spaces, you can 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.

See *SAS/ACCESS Interface to PC Files for SAS Viya: Reference* for more information about PC file formats.

Examples "Example 1: Exporting to a Delimited External Data Source" on page 104

"Example 2: Exporting a Subset of Observations to a CSV File" on page 110

OUTTABLE="*tablename*"

specifies the table name of the output 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.

Requirements You must have a license for SAS/ACCESS Interface to PC Files to export to a DBMS table.

When you export a DBMS table, you must specify the DBMS option.

Note Use OUTTABLE= for Microsoft Access database files.

Optional Arguments

DBMS=*identifier*

specifies the type of data to export. To export to a DBMS table, you must specify the DBMS option by using a valid database identifier. For DBMS=DLM, the default delimiter character is a space. However, you can use DELIMITER=*'char'*.

The following values are valid for the DBMS identifier.

Table 7.1 *DBMS Identifiers Supported in SAS*

Identifier	Output Data Source	Extension
CSV	Delimited file (comma-separated values)	.csv
DLM	Delimited file (default delimiter is a blank)	
JMP	JMP files, Version 7 or later format	.jmp
TAB	Delimited file (tab-delimited values)	.txt

Restriction The availability of an output external data source depends on these conditions:

- the operating environment and, in some cases, the platform as specified in the previous table.
- whether your site has a license for SAS/ACCESS Interface to PC Files. If you do not have a license, only delimited and JMP files are available.

See *SAS/ACCESS Interface to PC Files for SAS Viya: Reference* for a list of additional DBMS identifiers when using SAS/ACCESS Interface to PC Files.

Example "Example 1: Exporting to a Delimited External Data Source" on page 104

LABEL

specifies a variable label name. SAS writes these to the exported table as column names. If the label names do not already exist, SAS writes them to the exported table.

REPLACE

overwrites an existing file. If you do not specify REPLACE, the EXPORT procedure does not overwrite an existing file.

Example "Example 2: Exporting a Subset of Observations to a CSV File" on page 110

(*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=, PW=, READ=, or WRITE= data set options. To export a subset of data that meets a specified condition, you can use the WHERE option. For information about SAS data set options, see *SAS Viya Data Set Options: Reference*.

Example "Example 2: Exporting a Subset of Observations to a CSV File" on page 110

DBENCODING Statement

Indicates the encoding used to save data in JMP files.

Interaction: The DBENCODING statement is valid only when DBMS=JMP.

Syntax

DBENCODING=*12-char SAS encoding-value*;

Required Argument

12-char SAS encoding-value
indicates the encoding used to save data in JMP 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. This value can have a maximum length of 12 characters.

DELIMITER Statement

Specifies the delimiter to separate columns of data in the output file.

Default: Blank space

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

Example: "Example 1: Exporting to a Delimited External Data Source" on page 104

Syntax

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

Required Argument

***char* | '*nn*'x**
specifies the delimiter to use to separate values in the output file. You can specify the delimiter as a single character or as a hexadecimal value. For example, if you want columns of data to be separated by an ampersand, specify DELIMITER='&'.

FMTLIB Statement

Write SAS format values defined in the format catalog to the JMP file for the value labels.

Interaction: The FMTLIB statement is valid only when DBMS=JMP.

Syntax

FMTLIB=*<libref> format-catalog*;

Required Argument

<libref.> format-catalog
specifies the format catalog to be written to the JMP file.

PUTNAMES Statement

Writes SAS variable names as column headings to the first row of the exported data file.

Default: YES

Restriction: Valid only for the EXPORT procedure.

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

Example: "Example 3: Exporting to a Tab Delimited File with the PUTNAMES= Statement" on page 113

Syntax

PUTNAMES=YES | NO;

Required Arguments

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

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

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

Examples: EXPORT Procedure

Example 1: Exporting to a Delimited External Data Source

Features: PROC EXPORT statement options
DATA=
DBMS=
OUTFILE=
REPLACE

Other features: DELIMITER= statement

Details

This example exports the SASHelp.Class data set to a delimited external file. The following example is the SASHelp.Class data set before it is exported:

Output 7.1 *PROC PRINT of SASHelp.Class*

The SAS System

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

Program

```
proc export data=sashelp.class
   outfile="/userid/pathname/class"
   dbms=dlm replace;
```

```
    delimiter='&';
  run;
```

Program Description

Specify the input data set. Note that the filename does not contain an extension. DBMS=DLM specifies that the output file is a delimited file.

```
proc export data=sashelp.class
   outfile="/userid/pathname/class"
   dbms=dlm replace;
```

The DELIMITER option specifies that an & (ampersand) will delimit data fields in the output file.

```
    delimiter='&';
  run;
```

Log Examples

This SAS log displays this information about the successful export, including the generated SAS DATA step. The log is divided into sections only for documentation appearances.

Log 7.1 *SAS Log of Creating a Delimited File*

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
55
56         PROC PRINT DATA=sashelp.class;
57         RUN;

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.17 seconds
      cpu time            0.05 seconds



58
59         PROC EXPORT DATA=sashelp.class
60            OUTFILE="/userid/pathname/class.txt"
61                DBMS=TAB REPLACE;
62            PUTNAMES=YES;
63         RUN;

NOTE: Unable to open parameter catalog: SASUSER.PARMS.PARMS.SLIST in update mode. Temporary parameter
values will be saved to
WORK.PARMS.PARMS.SLIST.
64          /*********************************************************************
65          *   PRODUCT:   SAS
66          *   VERSION:   V.03.01
67          *   CREATOR:   External File Interface
68          *   DATE:      02SEP16
69          *   DESC:      Generated SAS Datastep Code
70          *   TEMPLATE SOURCE:  (None Specified.)
71          *********************************************************************/
72             data _null_;
73             %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
74             %let _EFIREC_ = 0;     /* clear export record count macro variable */
75             file '/userid/pathname/class.txt' delimiter='09'x DSD DROPOVER lrecl=32767;
76             if _n_ = 1 then        /* write column names or labels */
```

```
 77              do;
 78                put
 79                   "Name"
 80                '09'x
 81                   "Sex"
 82                '09'x
 83                   "Age"
 84                '09'x
 85                   "Height"
 86                '09'x
 87                   "Weight"
 88                ;
 89              end;
 90            set  SASHELP.CLASS   end=EFIEOD;
 91                format Name $8. ;
 92                format Sex $1. ;
 93                format Age best12. ;
 94                format Height best12. ;
 95                format Weight best12. ;
 96              do;
 97                EFIOUT + 1;
 98                put Name $ @;
 99                put Sex $ @;
 100               put Age @;
 101               put Height @;
 102               put Weight ;
 103               ;
 104             end;
 105            if _ERROR_ then call symputx('_EFIERR_',1);
                /* set ERROR detection macro variable */
 106            if EFIEOD then call symputx('_EFIREC_',EFIOUT);
 107            run;

 NOTE: The file '/userid/pathname/class.txt' is:
       Filename=/userid/pathname/class.txt,
       Owner Name=userid,Group Name=unix_pubs,
       Access Permission=-rwx------,
       Last Modified=02Sep2016:08:45:33

 NOTE: 20 records were written to the file '/userid/pathname/class.txt'.
       The minimum record length was 17.
       The maximum record length was 26.
 NOTE: There were 19 observations read from the data set SASHELP.CLASS.
 NOTE: DATA statement used (Total process time):
       real time           0.02 seconds
       cpu time            0.00 seconds


 19 records created in /userid/pathname/class.txt from SASHELP.CLASS.

NOTE: "/userid/pathname/class.txt" file was successfully created.
 NOTE: PROCEDURE EXPORT used (Total process time):
       real time           0.09 seconds
       cpu time            0.04 seconds


 108        PROC PRINT;
 109        RUN;
NOTE: There were 1 observations read from the data set WORK._GET_CAS_URL_TEMP_.
 NOTE: PROCEDURE PRINT used (Total process time):
       real time           0.02 seconds
       cpu time            0.01 seconds
```

```
 110
 111        PROC EXPORT DATA=sashelp.class
 112           OUTFILE="/userid/pathname/class.txt"
 113              DBMS=TAB REPLACE;
 114           PUTNAMES=NO;
 115        RUN;

 NOTE: Unable to open parameter catalog: SASUSER.PARMS.PARMS.SLIST in update mode. Temporary parameter
values will be saved to
 WORK.PARMS.PARMS.SLIST.
 116         /*********************************************************************
 117         *   PRODUCT:   SAS
 118         *   VERSION:   V.03.01
 119         *   CREATOR:   External File Interface
 120         *   DATE:      02SEP16
 121         *   DESC:      Generated SAS Datastep Code
 122         *   TEMPLATE SOURCE:  (None Specified.)
 123         *********************************************************************/
 124            data _null_;
 125            %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
 126            %let _EFIREC_ = 0;     /* clear export record count macro variable */
 127            file '/userid/pathname/class.txt' delimiter='09'x DSD DROPOVER lrecl=32767;
 128           set  SASHELP.CLASS   end=EFIEOD;
 129              format Name $8. ;
 130              format Sex $1. ;
 131              format Age best12. ;
 132              format Height best12. ;
 133              format Weight best12. ;
 134            do;
 135              EFIOUT + 1;
 136              put Name $ @;
 137              put Sex $ @;
 138              put Age @;
 139              put Height @;
 140              put Weight ;
 141              ;
 142            end;
 143            if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
 144            if EFIEOD then call symputx('_EFIREC_',EFIOUT);
 145            run;
 NOTE: The file '/userid/pathname/class.txt' is:
       Filename=/userid/pathname/class.txt.txt,
       Owner Name=userid,Group Name=unix_pubs,
       Access Permission=-rwx------,
       Last Modified=02Sep2016:08:45:33

 NOTE: 19 records were written to the file '/userid/pathname/class.txt'.

       The minimum record length was 17.
       The maximum record length was 21.
 NOTE: There were 19 observations read from the data set SASHELP.CLASS.
 NOTE: DATA statement used (Total process time):
       real time           0.01 seconds
       cpu time            0.01 seconds


 19 records created in /userid/pathname/class.txt from SASHELP.CLASS.
```

```
NOTE: "/userid/pathname/class.txt" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           0.08 seconds
      cpu time            0.04 seconds


146
147        PROC PRINT;
148        RUN;

NOTE: There were 1 observations read from the data set WORK._GET_CAS_URL_TEMP_.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds


149
150
151
152        OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```

Output Examples

The EXPORT procedure produces this external file:

Output 7.2 *External File*

```
Name&Sex&Age&Height&Weight
Alfred&M&14&69&112.5
Alice&F&13&56.5&84
Barbara&F&13&65.3&98
Carol&F&14&62.8&102.5
Henry&M&14&63.5&102.5
James&M&12&57.3&83
Jane&F&12&59.8&84.5
Janet&F&15&62.5&112.5
Jeffrey&M&13&62.5&84
John&M&12&59&99.5
Joyce&F&11&51.3&50.5
Judy&F&14&64.3&90
Louise&F&12&56.3&77
Mary&F&15&66.5&112
Philip&M&16&72&150
Robert&M&12&64.8&128
Ronald&M&15&67&133
Thomas&M&11&57.5&85
William&M&15&66.5&112
```

Example 2: Exporting a Subset of Observations to a CSV File

Features: PROC EXPORT statement options
- DATA=
- DBMS=
- OUTFILE=
- REPLACE

Details

This example exports the SAS data set SASHelp.Class to a delimited file.

Program

```
proc export data=sashelp.class (where=(sex='F'))
     outfile="/userid/pathname/Femalelist.csv"
     dbms=csv
     replace;
run;
```

Program Description

Specify the data set to be exported. The WHERE option requests a subset of the observations. The OUTFILE= option specifies the output file. The DBMS= option specifies that the output file is a CSV file, and overwrites the target CSV, if it exists.

```
proc export data=sashelp.class (where=(sex='F'))
     outfile="/userid/pathname/Femalelist.csv"
     dbms=csv
     replace;
run;
```

Log Examples

This SAS log displays this information about the successful export, including the generated SAS DATA step. The log is divided into sections only for documentation appearances.

Log 7.2 *Exporting to a CSV File*

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
55
56         proc export data=sashelp.class (where=(sex='F'))
57              outfile="/pathname/Femalelist.csv"
58              dbms=csv
59              replace;
60         run;

61          /*********************************************************************
62          *   PRODUCT:   SAS
63          *   VERSION:   V.03.01
64          *   CREATOR:   External File Interface
65          *   DATE:      02SEP16
66          *   DESC:      Generated SAS Datastep Code
67          *   TEMPLATE SOURCE:  (None Specified.)
68          *********************************************************************/
69             data _null_;
70             %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
71             %let _EFIREC_ = 0;     /* clear export record count macro variable */
72             file '/pathname/Femalelist.csv' delimiter=',' DSD DROPOVER lrecl=32767;
73             if _n_ = 1 then        /* write column names or labels */
74              do;
75                put
76                   "Name"
77                ','
78                   "Sex"
79                ','
80                   "Age"
81                ','
82                   "Height"
83                ','
84                   "Weight"
85                ;
86              end;
87            set  SASHELP.CLASS(where=(sex='F'))   end=EFIEOD;
88                format Name $8. ;
89                format Sex $1. ;
90                format Age best12. ;
91                format Height best12. ;
92                format Weight best12. ;
93              do;
94                EFIOUT + 1;
95                put Name $ @;
96                put Sex $ @;
97                put Age @;
98                put Height @;
99                put Weight ;
100               ;
101             end;
102            if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
103            if EFIEOD then call symputx('_EFIREC_',EFIOUT);
104            run;
```

```
NOTE: The file '/pathname/Femalelist.csv' is:
      Filename=/pathname/Femalelist.csv,
      Owner Name=userid,Group Name=unix_pubs,
      Access Permission=-rwx------,
      Last Modified=02Sep2016:14:16:19

NOTE: 10 records were written to the file '/pathname/Femalelist.csv'.
      The minimum record length was 17.
      The maximum record length was 26.
NOTE: There were 9 observations read from the data set SASHELP.CLASS.
      WHERE sex='F';
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.02 seconds


9 records created in /pathname/Femalelist.csv from SASHELP.CLASS.


NOTE: "/pathname/Femalelist.csv" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           0.10 seconds
      cpu time            0.04 seconds


105
106        OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```

Output Examples

The EXPORT procedure produces this external CSV file:

Output 7.3 *CSV File*

Name	Sex	Age	Height	Weight
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112

Example 3: Exporting to a Tab Delimited File with the PUTNAMES= Statement

Features: PROC EXPORT statement options
- DATA=
- DBMS=
- OUTFILE=
- PUTNAMES=
- REPLACE

Details

This example shows the export of a SAS data set, WORK.INVOICE, to a tab-delimited file. The first program uses PROC EXPORT with the PUTNAMES= statement and the second program does not. They show how the use of this statement affects column headings in a tab-delimited file.

The following display shows the SAS data set, WORK.INVOICE, before it is exported to a tab-delimited file:

Output 7.4 *PROC PRINT of WORK.INVOICE*

Obs	Invoice_ID	Billed_To	Amt_Billed	Country	Amt_in_US	Billed_By	Billed_On	Paid_On
1	11270	39045213	873600640	Brazil	3675848666.2	23915	05OCT2015	18OCT2015
2	11271	18543489	11063836	USA	1163836	457232	05OCT2015	.
3	11273	19783482	252148.5	USA	252148.5	239185	08OCT2015	11NOV2015
4	11276	14324742	1934460	USA	1934460	135673	06OCT2015	20OCT2015
5	11278	14898029	1400825	USA	1400825	239185	06OCT2015	19OCT2015
6	11280	39045213	8738600640	Brazil	3685848666.2	423286	06OCT2015	20OCT2015
7	11282	19783482	252148.5	USA	252148.5	457232	07OCT2015	25OCT2015
8	11285	38763919	2234301.3	Argentina	772847.05	23915	10OCT2015	30NOV2015
9	11286	43459747	14836604.06	Australia	11366352.06	423286	10OCT2015	.
10	11287	15432147	252148.5	USA	252148.5	457232	11OCT2015	04NOV2015
11	12051	39045213	8738600640	Brazil	3685848666.2	457232	02NOV2015	.
12	12102	18543489	11053836	USA	11053836	239185	17NOV2015	.
13	12263	19783482	252148.5	USA	252148.5	423286	05DEC2015	.
14	12468	14898029	1400825	USA	1400825	135873	24DEC2015	02JAN2016
15	12471	39045213	87386000640	Brazil	3675848666.2	457232	27DEC2015	.
16	12476	38763919	2234301.3	Argentina	772847.05	135873	24DEC2015	.
17	12478	15432147	252148.5	USA	252148.5	423286	24DEC2015	02JAN2016

Program

```
PROC PRINT DATA=WORK.INVOICE;
RUN;

PROC EXPORT DATA=WORK.INVOICE
   OUTFILE="/userid/pathname/invoice_names.txt"
       DBMS=TAB REPLACE;
   PUTNAMES=YES;
RUN;
PROC PRINT;
RUN;

PROC EXPORT DATA=WORK.INVOICE
   OUTFILE="/userid/pathname/invoice_names.txt"
      DBMS=TAB REPLACE;
   PUTNAMES=NO;
RUN;

PROC PRINT;
RUN;
```

Program Description

Use the PUTNAMES=YES statement in the EXPORT procedure. After WORK.INVOICE is printed, using the PUTNAMES=YES statement writes the SAS variables names as column names to the first row of the exported delimited file, Invoice_names.txt. The first row of data is then written to the second row of the delimited file.

```
PROC PRINT DATA=WORK.INVOICE;
RUN;

PROC EXPORT DATA=WORK.INVOICE
   OUTFILE="/userid/pathname/invoice_names.txt"
       DBMS=TAB REPLACE;
   PUTNAMES=YES;
RUN;
PROC PRINT;
RUN;
```

Impact of the PUTNAMES=NO statement. When you set this statement to NO, PROC EXPORT writes the first row of data to the first row of the exported delimited file. Therefore, the SAS variable names are skipped, and the columns are left unlabeled.

```
PROC EXPORT DATA=WORK.INVOICE
   OUTFILE="/userid/pathname/invoice_names.txt"
      DBMS=TAB REPLACE;
   PUTNAMES=NO;
RUN;

PROC PRINT;
RUN;
```

Log Examples

This SAS log displays information about the successful export, including the generated SAS DATA step. The log is divided into sections only for documentation appearances.

Log 7.3 *Exporting with the PUTNAMES Statement*

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
  56
 NOTE: Libref MYCAS was successfully assigned as follows:
       Engine:        CAS
       Physical Name: 19801cc2-65c1-4742-acfb-74ee86221874
 57
 58         PROC PRINT DATA=WORK.INVOICE;
 59         RUN;

 NOTE: There were 17 observations read from the data set WORK.INVOICE.
 NOTE: PROCEDURE PRINT used (Total process time):
       real time           0.26 seconds
       cpu time            0.07 seconds



 60
 61         PROC EXPORT DATA=WORK.INVOICE
 62              OUTFILE="/userid/pathname/MyDocuments/invoice_names.txt"
 63                DBMS=TAB REPLACE;
 64            PUTNAMES=YES;
 65         RUN;

 NOTE: Unable to open parameter catalog: SASUSER.PARMS.PARMS.SLIST in update mode. Temporary parameter
values will be saved to
 WORK.PARMS.PARMS.SLIST.
 66          /*********************************************************************
 67          *   PRODUCT:   SAS
 68          *   VERSION:   V.03.01
 69          *   CREATOR:   External File Interface
 70          *   DATE:      29MAR16
 71          *   DESC:      Generated SAS Datastep Code
 72          *   TEMPLATE SOURCE:  (None Specified.)
 73          *********************************************************************/
 74             data _null_;
 75             %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
 76             %let _EFIREC_ = 0;     /* clear export record count macro variable */
 77             file '/userid/pathname/MyDocuments/invoice_names.txt' delimiter='09'x DSD DROPOVER
lrecl=32767;
 78             if _n_ = 1 then        /* write column names or labels */
 79              do;
 80                put
 81                   "Invoice_ID"
 82                '09'x
 83                   "Billed_To"
 84                '09'x
 85                   "Amount_Billed_in_Local_Currency"
 86                '09'x
 87                   "Country"
 88                '09'x
 89                   "Amount_Billed_in_US_Dollars"
 90                '09'x
 91                   "Billed_By"
 92                '09'x
 93                   "Billed_On"
 94                '09'x
 95                   "Paid_On"
 96                ;
 97              end;
```

```
98            set  WORK.INVOICE   end=EFIEOD;
99                format Invoice_ID best12. ;
100               format Billed_To best12. ;
101               format Amount_Billed_in_Local_Currency best12. ;
102               format Country $9. ;
103               format Amount_Billed_in_US_Dollars best12. ;
104               format Billed_By best12. ;
105               format Billed_On $9. ;
106               format Paid_On date9. ;
107             do;
108               EFIOUT + 1;
109               put Invoice_ID @;
110               put Billed_To @;
111               put Amount_Billed_in_Local_Currency @;
112               put Country $ @;
113               put Amount_Billed_in_US_Dollars @;
114               put Billed_By @;
115               put Billed_On $ @;
116               put Paid_On ;
117               ;
118             end;
119            if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
120            if EFIEOD then call symputx('_EFIREC_',EFIOUT);
121            run;

NOTE: The file '/userid/pathname/MyDocuments/invoice_names.txt' is:
      Filename=/userid/pathname/MyDocuments/invoice_names.txt,
      Owner Name=userid,Group Name=unix_pubs,
      Access Permission=-rwx------,
      Last Modified=02Sep2016:16:34:34

NOTE: 18 records were written to the file '/userid/pathname/MyDocuments/invoice_names.txt'.
      The minimum record length was 53.
      The maximum record length was 116.
NOTE: There were 17 observations read from the data set WORK.INVOICE.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds


17 records created in /userid/pathname/MyDocuments/invoice_names.txt from WORK.INVOICE.


NOTE: "/userid/pathname/MyDocuments/invoice_names.txt" file was successfully created.
NOTE: PROCEDURE EXPORT used (Total process time):
      real time           0.11 seconds
      cpu time            0.03 seconds


122        PROC PRINT;
123        RUN;

NOTE: There were 17 observations read from the data set WORK.INVOICE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.21 seconds
      cpu time            0.06 seconds
```

```
 124
 125        PROC EXPORT DATA=WORK.INVOICE
 126           OUTFILE="/userid/pathname/MyDocuments/invoice_data_1st.txt"
 127              DBMS=TAB REPLACE;
 128           PUTNAMES=NO;
 129        RUN;

 NOTE: Unable to open parameter catalog: SASUSER.PARMS.PARMS.SLIST in update mode. Temporary parameter
values will be saved to
 WORK.PARMS.PARMS.SLIST.
 130         /*********************************************************************
 131         *   PRODUCT:   SAS
 132         *   VERSION:   V.03.01
 133         *   CREATOR:   External File Interface
 134         *   DATE:      29MAR16
 135         *   DESC:      Generated SAS Datastep Code
 136         *   TEMPLATE SOURCE:  (None Specified.)
 137         *********************************************************************/
 138            data _null_;
 139            %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
 140            %let _EFIREC_ = 0;     /* clear export record count macro variable */
 141            file '/userid/pathname/MyDocuments/invoice_data_1st.txt' delimiter='09'x DSD DROPOVER
lrecl=32767;
 142           set  WORK.INVOICE   end=EFIEOD;
 143               format Invoice_ID best12. ;
 144               format Billed_To best12. ;
 145               format Amount_Billed_in_Local_Currency best12. ;
 146               format Country $9. ;
 147               format Amount_Billed_in_US_Dollars best12. ;
 148               format Billed_By best12. ;
 149               format Billed_On $9. ;
 150               format Paid_On date9. ;
 151             do;
 152               EFIOUT + 1;
 153               put Invoice_ID @;
 154               put Billed_To @;
 155               put Amount_Billed_in_Local_Currency @;
 156               put Country $ @;
 157               put Amount_Billed_in_US_Dollars @;
 158               put Billed_By @;
 159               put Billed_On $ @;
 160               put Paid_On ;
 161               ;
 162             end;
```

```
163            if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
 164            if EFIEOD then call symputx('_EFIREC_',EFIOUT);
 165            run;

 NOTE: The file '/userid/pathname/MyDocuments/invoice_data_1st.txt' is:
       Filename=/userid/pathname/MyDocuments/invoice_data_1st.txt,
       Owner Name=userid,Group Name=unix_pubs,
       Access Permission=-rwx------,
       Last Modified=02Sep2016:16:34:34

 NOTE: 17 records were written to the file '/userid/pathname/MyDocuments/invoice_data_1st.txt'.
       The minimum record length was 53.
       The maximum record length was 72.
 NOTE: There were 17 observations read from the data set WORK.INVOICE.
 NOTE: DATA statement used (Total process time):
       real time           0.02 seconds
       cpu time            0.01 seconds


 17 records created in /userid/pathname/MyDocuments/invoice_data_1st.txt from WORK.INVOICE.


 NOTE: "/userid/pathname/MyDocuments/invoice_data_1st.txt" file was successfully created.
 NOTE: PROCEDURE EXPORT used (Total process time):
       real time           0.12 seconds
       cpu time            0.04 seconds


 166
 167        PROC PRINT;
 168        RUN;

 NOTE: There were 17 observations read from the data set WORK.INVOICE.
 NOTE: PROCEDURE PRINT used (Total process time):
       real time           0.21 seconds
       cpu time            0.05 seconds



 169
 170
 171        OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```

Chapter 8

FEDSQL Procedure

Overview: FEDSQL Procedure

The FEDSQL procedure enables you to submit FedSQL language statements from a Base SAS session. The FedSQL language is the SAS implementation of the ANSI SQL: 1999 core standard. It provides support for extended data types, such as DECIMAL, INTEGER, and VARCHAR, and other ANSI 1999 core compliance features and proprietary extensions. FedSQL provides data access technology that brings a scalable, threaded, high-performance way to access, manage, and share relational data in multiple data sources. For applications, FedSQL provides a common SQL syntax across all of the data sources that it supports. That is, FedSQL is a vendor-neutral SQL dialect that accesses data from various data sources without having to submit queries in the SQL dialect that is specific to the data source. When possible, FedSQL queries are optimized with multi-threaded algorithms to resolve large-scale operations.

In SAS Viya, PROC FEDSQL provides support for two data sources: Base SAS data sets and SAS Cloud Analytic Services tables. In its initial release for SAS Cloud Analytic

Services, FedSQL can be used to create in-memory CAS output tables from CAS input tables. The input data must previously have been loaded into the SAS Cloud Analytic Services server. You cannot join SAS data sets and CAS tables.

For information about the FedSQL language statements that are supported for SAS data sets, see SAS 9.4 FedSQL Language: Reference, Fourth Edition. For information about the FedSQL language statements supported for SAS Cloud Analytic Services tables, see *SAS FedSQL Language Reference for SAS Cloud Analytic Services*.

Concepts: FEDSQL Procedure

Benefits of FedSQL

FedSQL provides the following benefits:

- FedSQL conforms to the SQL 1999 ANSI standard. This conformance allows it to process queries in its own language as well as the native languages of other DBMSs that conform to the ANSI 1999 standard.
- FedSQL supports implicit pass-through, which is the process of translating query code into equivalent data source-specific code, so that it can be passed directly to the data source for processing. Implicit pass-through improves query response time and enhances security. The volume of data being transferred is reduced by performing the query on the data source. The number of rows that are transferred from the data source to FedSQL can be significantly reduced, thereby decreasing the overall query processing time. The leveraging of data source-specific capabilities, such as massively parallel processing, are specific to a data source. Other examples of special capabilities are advanced join techniques, data partitioning, table statistics, and column statistics. These capabilities often allow the data source to perform the SQL query more quickly than FedSQL.

Data Source Support

In SAS Viya, PROC FEDSQL can access the following data sources:

- SAS data sets
- SAS Cloud Analytic Services tables

Syntax: FEDSQL Procedure

PROC FEDSQL <*option(s)*>;
…*FedSQL statements*
QUIT;

Statement	Task	Example
PROC FEDSQL	Specify that the subsequent input is FedSQL statements.	Ex. 1, Ex. 2, Ex. 3, Ex. 4

Statement	Task	Example
QUIT	Stop the execution of the FEDSQL procedure.	Ex. 1, Ex. 2, Ex. 3, Ex. 4

PROC FEDSQL Statement

Specifies that the subsequent input is FedSQL statements

Restriction: You cannot overwrite an existing table. You must drop the table by using the DROP TABLE statement and re-create the table with the CREATE TABLE statement. When working with CAS tables, you can use the REPLACE= table option to overwrite an existing table of the same name.

Interaction: The procedure processes nonexistent values as SAS missing values.

Examples:
"Example 1: Creating a SAS Data Set" on page 136
"Example 2: Creating a SAS Data Set from Existing Tables" on page 140
"Example 3: Loading and Joining Tables in CAS" on page 141
"Example 4: Obtain Query Details with the _METHOD= Option" on page 145

Syntax

PROC FEDSQL <*SAS-connection-option* | *CAS-connection-option*><*processing-options*>;

Summary of Optional Arguments

CAS Connection

SESSREF=*session-name*
: specifies to run the FedSQL statements in a SAS Cloud Analytic Services (CAS) session. The CAS session is identified by its session name.

SESSUUID="*session-uuid*"
: specifies to run the FedSQL statements in a SAS Cloud Analytic Services (CAS) session. The CAS session is identified by its universally unique identifier (UUID).

General Processing

_METHOD= TRUE | FALSE
: prints a text description of the FedSQL query plan.

_POSTOPTPLAN=TRUE | FALSE
: prints an XML tree illustrating the FedSQL query plan.

ERRORSTOP | NOERRORSTOP
: specifies whether the procedure stops executing if it encounters an error.

LABEL | NOLABEL
: specifies whether to use the column label or the column name as the column heading.

MEMSIZE=*n* | *n*M | *n*G

NOPRINT
: suppresses the normal display of results.

NUMBER
specifies to include a column named Row, which is the row (observation) number of the data as the rows are retrieved.

STIMER
specifies to write a subset of system performance statistics, such as time-elapsed statistics, to the SAS log.

XCODE=ERROR | WARNING | IGNORE
controls the behavior of the SAS session when an NLS transcoding failure occurs.

SAS Connection

LIBS=*libref* | (*libref1 libref2 ...librefn*)
restricts the data source connection to the specified libref(s).

NOLIBS CONN="*connection-string*"
overrides the default data source connection with a specified connection string.

Optional Arguments

_METHOD= TRUE | FALSE
prints a text description of the FedSQL query plan. A FedSQL query is broken into stages. Each stage of execution requires a standalone SQL query. This option generates a text description of the number and types of stages in the query plan. The information is written to the SAS log.

Default FALSE

See "Example 4: Obtain Query Details with the _METHOD= Option" on page 145

_POSTOPTPLAN=TRUE | FALSE
prints an XML tree illustrating the FedSQL query plan. A FedSQL query is broken into stages. Each stage of execution requires a standalone SQL query. This option generates an XML tree that illustrates each stage of the FedSQL query plan and the results from each execution stage. The information is written to the SAS log.

Default FALSE

Note The XML tree can be very long.

ERRORSTOP | NOERRORSTOP
specifies whether the procedure stops executing if it encounters an error. In a batch or noninteractive session, ERRORSTOP instructs the procedure to stop executing the statements but to continue checking the syntax after it has encountered an error. NOERRORSTOP instructs the procedure to execute the statements and to continue checking the syntax after an error occurs.

Default NOERRORSTOP in an interactive SAS session; ERRORSTOP in a batch or noninteractive session

Tips ERRORSTOP has an effect only when SAS is running in the batch or noninteractive execution mode.

NOERRORSTOP is useful if you want a batch job to continue executing SQL procedure statements after an error is encountered.

LABEL | NOLABEL

specifies whether to use the column label or the column name as the column heading.

Default LABEL

Interactions If a column does not have a label, the procedure uses the column's name as the column heading.

A column alias overwrites the label or column name as the column heading.

LIBS=*libref* | (*libref1 libref2 ...librefn*)

restricts the data source connection to the specified libref(s). By default, PROC FEDSQL builds a data source connection that includes all currently assigned librefs to pass to the FedSQL program. When the LIBS= option is specified, only the specified librefs are included in the data source connection; all other librefs are ignored.

When you specify a list of librefs, the order of the list defines the library order. The Work library is implicitly included as the first library in all lists and is the default library for data references that do not specify a libref.

Alias LIBNAMES=

Restriction LIBS= cannot be used with SESSREF= (or SESSUID=).

Interactions The LIBS= option accomplishes the same result as the NOLIBS CONN= option. The use of LIBS= is recommended over NOLIBS CONN= because you do not need to know how to write a data source connection string to use LIBS=.

If both LIBS= and NOLIBS CONN= are specified, the last option on the procedure statement is applied.

Tip If you are curious about how LIBS= affects library assignments, set the MSGLEVEL=i system option before running a PROC FEDSQL request with LIBS=. The option will produce Include and Ignore messages for each of the LIBNAME statements that are processed in the procedure request.

Example The following PROC FEDSQL procedure statement specifies to create a data source connection that uses only librefs MyLib3 and MyLib4:

```
proc fedsql libs=(mylib3 mylib4);
```

MEMSIZE=*n* | *n*M | *n*G

specifies a limit for the amount of memory that is used for an underlying query (such as a SELECT statement), so that allocated memory is available to support other PROC FEDSQL operations. Specify the memory limit in multiples of 1 (bytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). For example, the value 23m specifies 24,117,248 bytes of memory. In CAS, MEMSIZE specifies the memory for a single CAS worker.

Alias STMTMEMLIMIT=

Tip Generally, specifying a memory limit is not necessary unless FedSQL reports a memory problem error.

NOLIBS CONN="*connection-string*"
overrides the default data source connection with a specified connection string. By default, PROC FEDSQL builds a data source connection that includes all currently assigned librefs to pass to the FedSQL program. NOLIBS turns off the default data source connection. CONN= specifies an alternate data source connection string.

Connection-string can have the following attributes. The attributes are data source dependent.

CATALOG="*catalog-identifier*";
specifies an arbitrary identifier for an SQL catalog, which groups logically related schemas. A catalog name can be up to 32 characters long.

DRIVER="*driver-name*";
specifies the data source that you want to connect to.

SCHEMA=*value*
specifies a SCHEMA= in which to create or read data.

Value must be in the form:

```
(NAME=value; PRIMARYPATH=value)
```

NAME=*identifier*
specifies an arbitrary identifier for an SQL schema. Any identifier is valid (for example, name=myfiles). The schema identifier is an alias for the physical location of the SAS library, which is much like the Base SAS libref. A schema name must be a valid SAS name and can be up to 32 characters long. You must specify a schema identifier.

PRIMARYPATH={*pathname*}
specifies the physical location for the SAS library. In most operating environments, this is a directory path. You must specify a primary path.

Restriction
NOLIBS CONN= cannot be used with SESSREF= (or SESSUID=).

Interactions
NOLIBS CONN= accomplishes the same result as LIBS= on page 125. The use of LIBS= is recommended over NOLIBS CONN= because you do not need to know how to write a data source connection string to use LIBS=.

If both NOLIBS CONN= and LIBS= are specified, the last option on the procedure statement is applied.

Tip
Data source connection strings are difficult to write. If you set the MSGLEVEL=i system option and re-run a failed request, the data source connection strings that were generated by the procedure are written to the SAS log. Use the generated connection strings as a guideline.

Examples
Here is a PROC FEDSQL procedure statement that uses NOLIBS CONN= to specify a connection string for Base SAS data:

```
proc fedsql nolibs
   conn="driver=base;
         catalog=base;
         schema=(name=base;primarypath=.)";
```

Here is a procedure statement that specifies two paths:

```
proc fedsql nolibs
 conn="(driver=base;
catalog=work;
```

```
schema=(name=work;primarypath={%sysfunc(pathname(work))}));
(driver=base;
catalog=base;
schema=(name=base;primarypath='c:\temp\base'))";
```

NOPRINT
suppresses the normal display of results.

Interaction NOPRINT affects the value of the SQLOBS automatic macro variable, which contains the number of rows that are executed by a statement.

NUMBER
specifies to include a column named Row, which is the row (observation) number of the data as the rows are retrieved.

Default No row numbers.

Restriction This option is not supported with SESSREF= (or SESSUUID=).

SESSREF=*session-name*
specifies to run the FedSQL statements in a SAS Cloud Analytic Services (CAS) session. The CAS session is identified by its session name. By default, PROC FEDSQL builds a data source connection that includes all currently assigned librefs to pass to the FedSQL program. The SESSREF= option overrides the default behavior and runs the request in the specified CAS session instead. The CAS session must have been previously established by using the CAS statement. If the specified CAS session does not exist, the procedure terminates. You can connect to a single CAS session at a time.

Interactions Use SESSREF= or SESSUUID= to connect to SAS Cloud Analytic Services. If both options are specified, the last option in the procedure statement is applied.

If both SESSREF= and LIBS= (or NOLIBS CONN=) are specified on the procedure statement, SESSREF= is applied and the other option is ignored.

Example Here is a PROC FEDQL procedure statement that specifies SESSREF=

```
proc fedsql sessref=mysess;
```

SESSUUID="*session-uuid*"
specifies to run the FedSQL statements in a SAS Cloud Analytic Services (CAS) session. The CAS session is identified by its universally unique identifier (UUID). By default, PROC FEDSQL builds a data source connection that includes all currently assigned librefs to pass to the FedSQL program. The SESSUUID= option overrides the default behavior and runs the request in the specified CAS session instead. The CAS session must have been previously established by using the CAS statement. The CAS statement generates a UUID value. If the specified CAS session does not exist, the procedure terminates. You can connect to a single CAS session at a time.

Interactions Use SESSREF= or SESSUUID= to connect to SAS Cloud Analytic Services. If both options are specified, the last option in the procedure statement is applied.

If both SESSUUID= and LIBS= (or NOLIBS CONN=) are specified in the procedure statement, SESSUUID= is applied and the other option is ignored.

Example Here is a PROC FEDSQL procedure statement that specifies SESSUUID=:

```
proc fedsql sessuuid="76904741-fb09-554d-a8de-6cbce2a0e0e5";
```

The UUID value can be enclosed in single or double quotation marks.

STIMER
specifies to write a subset of system performance statistics, such as time-elapsed statistics, to the SAS log. When STIMER is in effect, the procedure writes to the SAS log a list of computer resources used for each step and the entire SAS session.

Default No performance statistics are written to the SAS log.

Interaction If the SAS system option FULLSTIMER is in effect, the complete list of computer resources is written to the SAS log.

XCODE=ERROR | WARNING | IGNORE
controls the behavior of the SAS session when an NLS transcoding failure occurs. Transcoding failures can occur during row input or output operations, or during string assignment. Transcoding is the process of converting character data from one encoding to another encoding.

ERROR
specifies that a run-time error occurs, which causes row processing to halt. An error message is written to the SAS log. This is the default behavior.

WARNING
specifies that the incompatible character is set to a substitution character. A warning message is written to the SAS log.

IGNORE
specifies that the incompatible character is set to a substitution character. No messages are written to the SAS log.

Default ERROR

QUIT Statement

Stops the execution of the FEDSQL procedure.

Interaction: Unlike other SAS procedures, PROC FEDSQL does not recognize step boundaries. That is, if you submit a DATA step or another procedure step without first specifying the QUIT statement, the FedSQL language issues a syntax error and PROC FEDSQL continues processing. The QUIT statement is required to stop the FEDSQL procedure.

Syntax

QUIT;

Details

When the FEDSQL procedure reaches the QUIT statement, all resources allocated by the procedure are released. You can no longer execute FedSQL language statements without invoking the procedure again. However, the connection to the data source server is not lost, because that connection was made through the LIBNAME statement. As a result, any subsequent invocation of the procedure that uses the same libref executes almost instantaneously because the LIBNAME engine is already connected to the server.

Using the FEDSQL Procedure

Data Source Connection

PROC FEDSQL can execute requests in a SAS session, or it can execute requests in a SAS Cloud Analytic Services session. By default, PROC FEDSQL connects to a data source by using SAS data access services. You can override this default behavior by specifying connection options. The FEDSQL procedure is not affected by the CASNAME= system option.

Understanding the Default Data Source Connection

PROC FEDSQL connects to a data source by using the attributes of currently assigned librefs. Attributes include the physical location of the data, and for some data sources, access information such as network information used to access the data server, and user identification and password.

You first submit the LIBNAME statement for a SAS engine and then submit PROC FEDSQL. The supported engine is SAS V9 BASE.

This example illustrates how PROC FEDSQL accesses a data source by using the attributes of a previously assigned libref. The LIBNAME statement assigns the libref MyFiles, specifies the BASE engine, and then specifies the physical location for the SAS data set. The FedSQL program then creates a SAS data set named MyFiles.Table1 at the location specified in the LIBNAME statement.

```
libname allfiles base 'C:\sharedfiles';
libname myfiles base 'C:\myfiles';

proc fedsql;
   create table myfiles.table1 (x double);
   insert into myfiles.table1 values (1.0);
   insert into myfiles.table1 values (2.0);
   insert into myfiles.table1 values (3.0);
quit;
```

In the example, two librefs are assigned in the SAS session: AllFiles and MyFiles. The procedure builds a data source connection string that includes all the active librefs in the SAS session and sends it to the FedSQL program. You reference a particular library by specifying its libref in a two-part table name in the form *libref.table-name*. If you do not specify a libref, the table is created in the SAS Work library.

The procedure uses libref attributes for connection information only (such as physical location). PROC FEDSQL does not use libref attributes that define behavior. For example, if a previously submitted LIBNAME statement for the BASE engine specifies

that SAS data sets are to be compressed, the compression attribute is not used by the procedure.

Note: PROC FEDSQL connects immediately, so an error is generated if the LIBNAME statement includes the DEFER=YES option.

Note: To access data created with the SAS 9.4 BASE engine, you must have SAS Viya Bridge software.

Connecting with LIBS=

When multiple librefs are active in the SAS session, you might want to include the LIBS= option in the PROC FEDSQL statement. LIBS= restricts the data source connection to the specified libref or librefs. The following example illustrates the use of the LIBS= option. In the example, the LIBS= option specifies to use only libref MyFiles.

```
libname allfiles base 'C:\sharedfiles';
libname myfiles base 'C:\myfiles';

proc fedsql libs=myfiles;
   create table myfiles.table1 (x double);
   insert into myfiles.table1 values (1.0);
   insert into myfiles.table1 values (2.0);
   insert into myfiles.table1 values (3.0);
quit;
```

When LIBS= is specified, the data source connection string generated by SAS data access services includes only information from the specified libref. For more information, see "LIBS=*libref* | (*libref1 libref2 ...librefn*)" on page 125.

NOLIBS CONN= accomplishes the same thing as LIBS=, except you must specify a data source connection string, which is more difficult to do.

For information about the FedSQL statements supported in the default connection and LIBS=, see SAS 9.4 FedSQL Language: Reference, Fourth Edition.

Connecting with SESSREF= or SESSUUID=

You connect to a SAS Cloud Analytic Services session by specifying the SESSREF= (or SESSUUID=) procedure option. When either of these options is specified, both the default connection mechanism and the LIBS= and NOLIBS CONN= options are ignored. Instead, the procedure connects to the specified CAS session.

You must first submit the CAS statement to establish a CAS session. To interact with data in a CAS session, you need a CAS library (caslib). Caslibs behave differently from librefs:

- Caslibs can be personal, pre-defined by an administrator, or added manually. Your level of authorization determines your interaction with each type of caslib.
- When you start a CAS session, the personal caslib CASUSER is automatically allocated and becomes the active caslib for your session, unless other caslibs are defined.
- You can determine what caslibs are available to you by using the CASLIB statement with the LIST argument. If you are authorized, you can add caslibs with the CASLIB statement.
- Pre-defined caslibs often provide pre-loaded data. For the other caslibs, you must create or load data in CAS before you can access the data using SAS Cloud Analytic Services. You can create data in CAS by using the DATA step or by using PROC

DS2. You can load existing data by using the CASUTIL procedure. The PROC CASUTIL LOAD statement supports loading of data from SAS, ODBC, Hadoop, Impala, Teradata, Oracle and PostGres into CAS.

After ensuring that the caslib that you want to access is the active caslib in your CAS session (and loading data if necessary), submit PROC FEDSQL. In the PROC FEDSQL statement, specify the session name that you defined in the CAS statement in the SESSREF= option. Or specify the session UUID in the SESSUUID= option.

The following example illustrates the use of the SESSREF= option.

```
cas mysess; 1

proc casutil; 2
     load data=sashelp.cars outcaslib="casuser"
 casout="demotable";
run;

proc fedsql sessref=mysess;
  create table cars2 as
  select make, model, origin from demotable;
quit;
```

1. The CAS statement establishes a CAS session named “MySess”. The CAS statement does not specify a CAS library. Therefore, the personal caslib, CASUSER, is used for the operation.
2. The CASUTIL procedure is used to load the SAS data set Sashelp.Cars into a CAS table named DemoTable. The OUTCASLIB= argument specifies the target caslib.
3. PROC FEDSQL is executed with the SESSREF= procedure option. SESSREF= specifies to connect to SAS Cloud Analytic Services, instead of using the default SAS connection. The procedure connects to the CAS session using the active caslib, CASUSER, and specifies to create a new table from loaded table DemoTable.

When only one caslib is active in a CAS session, there is no need to reference the caslib in the table name. Use a two-part name to reference your tables (casuser.demotable) when more than one caslib is active.

The CAS tables that you create with PROC FEDSQL are in-memory tables. That is, the tables are available for the duration of the CAS session and are accessible only to the current session. PROC FEDSQL does not provide a way to persist a table or to share the table with other CAS sessions. To persist or share a CAS output table, use the CASUTIL procedure.

Note: Although CAS tables are in-memory tables, PROC FEDSQL will not overwrite an existing table of the same name. Specify the REPLACE= table option to overwrite an existing table with a replacement table. Or, use the DROP TABLE statement to remove the initial table before creating the replacement table.

For more information, see:

- *SAS Cloud Analytic Services: Fundamentals*
- CAS statement, CASLIB statement, and CASUTIL procedure in *SAS Cloud Analytic Services: Language Reference*
- SESSREF= on page 127.
- SESSUUID= on page 127.

Using FedSQL in SAS Cloud Analytic Services

SAS Cloud Analytic Services is an alternative environment for processing FedSQL queries. When you specify the SESSREF= (or SESSUUID=) option, PROC FEDSQL executes your FedSQL query on a SAS Cloud Analytic Services (CAS) server. The CAS server is a Symmetric Multi-Processing (SMP) server. A FedSQL request executing in a CAS server can perform manipulations on multiple table rows concurrently using multiple threads and worker nodes, reducing the time required to process large tables.

The FedSQL functionality for SAS Cloud Analytic Services is a subset of the FedSQL capabilities available in SAS 9.4.

- The following FedSQL statements are supported in CAS:
  - CREATE TABLE, with the AS query expression
  - SELECT
  - DROP TABLE
- The following SELECT statement features are not supported:
  - SET operations
  - correlated subqueries
  - dictionary queries
  - views
  - DS2 user-defined functions (UDFs).

For an example of joining data in CAS, see "Example 3: Loading and Joining Tables in CAS" on page 141.

For more information about the FedSQL statements supported in CAS, see *SAS FedSQL Language Reference for SAS Cloud Analytic Services*.

Applying the FedSQL Table Options

When you access a data source with PROC FEDSQL, you can apply FedSQL table options in the subsequent FedSQL statements. A table option specifies actions that enable you to perform operations on a table such as assigning buffer page size or specifying passwords. A FedSQL table option performs much of the same functionality as a Base SAS data set option.

FedSQL table options are used to apply options when you access a data source within PROC FEDSQL. For example, the following code applies a table option to the SAS data set in order to specify the size of a permanent buffer page for the new table:

```
libname myfiles base 'C:\myfiles';

proc fedsql;
   create table myfiles.table1 {options bufsize=16k}(x double) ;
   insert into myfiles.table1 values (1.0);
   insert into myfiles.table1 values (2.0);
   insert into myfiles.table1 values (3.0);
quit;
```

For a list of table options for SAS data sets, see SAS 9.4 FedSQL Language: Reference, Fourth Edition. For a list of table options for CAS tables, see *SAS FedSQL Language Reference for SAS Cloud Analytic Services*.

Macro Variables

Using Macro Variables in a Literal String

Macro variables enable you to dynamically modify text in a program through symbolic substitution. When you reference a macro variable in a program, the macro processor replaces the reference with the value of the specified macro variable.

With PROC FEDSQL, you can use a macro variable on a subsequent FedSQL statement. However, if a macro variable occurs within a literal string, you cannot enclose the string in double quotation marks, which is required by the macro processor to resolve the macro variable reference. You cannot enclose the string in double quotation marks because FedSQL statements consider a string enclosed in double quotation marks to be a delimited (case sensitive) identifier such as a table or column name.

To reference a macro variable in a literal string, use the SAS macro function %TSLIT, which overrides the need for double quotation marks around the literal string and puts single quotation marks around the input value. For example, the following statement includes the %TSLIT function to specify the &SYSHOSTNAME macro variable, which returns the host name of the computer on which it is executed:

```
select %tslit(&syshostname);
```

The %TSLIT macro function is stored in the default autocall macro library. For more information, see “Referencing a Macro Variable in a Delimited Identifier” in the SAS 9.4 FedSQL Language: Reference, Fourth Edition.

Using Macro Variables Set by the Procedure

PROC FEDSQL sets up macro variables with certain values after it executes each statement. These macro variables can be tested inside a macro to determine whether to continue executing the PROC step. After each statement has executed, the following macro variable is updated with these values:

SQLRC
: contains the following status values that indicate the success of the PROC FEDSQL statement:

  0
  : PROC statement completed successfully with no errors.

  4
  : PROC statement encountered a situation for which it issued a warning. The statement continued to execute.

  8
  : PROC statement encountered an error. The statement stopped execution at this point.

  16
  : PROC statement encountered a run-time error. For example, this error code is used when a subquery (that can return only a single value) evaluates more than one row. These errors can be detected only during run time.

Security

SAS Viya supports a security scheme that is implemented by administrators. For more information, see *SAS Cloud Analytic Services: Authorization*.

In addition, PROC FEDSQL supports password protection for SAS data sets.

Base SAS software enables you to restrict access to SAS data sets by assigning SAS passwords to the files. You can specify three levels of protection: read, write, and alter.

With PROC FEDSQL, you assign or specify a password for a data source using the FedSQL table options ALTER=, PW=, READ=, and WRITE=. For example, the following code applies the FedSQL table option PW= in order to assign READ, WRITE, and ALTER passwords to a SAS data set:

```
libname myfiles base 'C:\myfiles';

proc fedsql;
   create table myfiles.table1 {options pw=luke}(x double) ;
   insert into myfiles.table1 values (1.0);
   insert into myfiles.table1 values (2.0);
   insert into myfiles.table1 values (3.0);
quit;
```

FedSQL table options perform much of the same functionality as Base SAS data set options. However, Base SAS data set options are not supported in the PROC FEDSQL statement. Therefore, FedSQL table options must be used to assign or specify passwords when accessing a data source with PROC FEDSQL.

A SAS password does not control access to a SAS file beyond the SAS System. You should use the operating system-supplied utilities and file-system security controls to control access to SAS files outside SAS. For more information about SAS passwords, see SAS 9.4 FedSQL Language: Reference, Fourth Edition.

FedSQL Data Type Support for SAS Data Sets

In PROC FEDSQL, when you submit FedSQL statements, all FedSQL language data types are supported. For information about the FedSQL data types, see SAS 9.4 FedSQL Language: Reference, Fourth Edition.

However, in a Base SAS session, when you are not submitting PROC FEDSQL, FedSQL data types are translated to and from predetermined legacy SAS data types, which are SAS numeric and SAS character. For example, when you submit the CONTENTS procedure on a table that is created with the FedSQL language, the DATE data type is reported as a SAS numeric. The following table lists the FedSQL data types and how they are translated to and from SAS data types:

Table 8.1 *FedSQL Data Type Translation*

FedSQL Data Type	Legacy SAS Data Type	Description
BIGINT	SAS numeric	Applies the SAS format 20. Because a SAS numeric is a DOUBLE, which is an approximate numeric data type rather than an exact numeric data type, there is potential for loss of precision.
BINARY(*n*)	SAS character	Applies the SAS format $*n*.
CHAR(*n*)	SAS character	Applies the SAS format $*n*.
DATE	SAS numeric	Applies the SAS format DATE9. Valid SAS date values are in the range from 1582-01-01 to 9999-12-31. Dates outside the SAS date range are not supported and are treated as invalid dates.
DECIMAL|NUMERIC(*p*,**s**)	SAS numeric	Applies the SAS format 11.2.
DOUBLE	SAS numeric	
FLOAT(*p*)	SAS numeric	
INTEGER	SAS numeric	Applies the SAS format 11.
NCHAR(*n*)	SAS character	Applies the SAS format $*n*.
NVARCHAR(*n*)	SAS character	Applies the SAS format $*n*.
REAL	SAS numeric	
SMALLINT	SAS numeric	Applies the SAS format 6.
TIME(*p*)	SAS numeric	Applies the SAS format TIME8.
TIMESTAMP(*p*)	SAS numeric	Applies the SAS format DATETIME19.2.
TINYINT	SAS numeric	Applies the SAS format 4.
VARBINARY(*n*)	SAS character	Applies the SAS format $*n*.
VARCHAR(*n*)	SAS character	Applies the SAS format $*n*.

FedSQL Data Type Support for CAS Tables

CAS tables support CHAR, DOUBLE, and VARCHAR data types. FedSQL supports the ability to create CAS output tables from CAS input tables only. Therefore, these are the only data types supported for CAS.

Table 8.2 *FedSQL Data Type Translation for CAS Tables*

FedSQL Data Type	CAS Data Type	Description
CHAR(*n*)	CHAR	Stores a fixed-length character string, where *n* is the maximum number of characters to store. The maximum number of characters is required to store each value regardless of the actual size of the value. If char(10) is specified and the character string is only five characters long, the value is right-padded with spaces.
DOUBLE	DOUBLE	Stores a signed, approximate, double-precision, floating-point number. Allows numbers of large magnitude and permits computations that require many digits of precision to the right of the decimal point. For SAS Cloud Analytic Services, this is a 64-bit double-precision, floating-point number.
VARCHAR(*n*)	VARCHAR	Stores a varying-length character string.

Date, time, and timestamp values are supported as DOUBLEs in CAS tables.

Examples: FEDSQL Procedure

Example 1: Creating a SAS Data Set

Features: PROC FEDSQL statement
QUIT statement

Other features: LIBNAME statement
FedSQL language statements
PROC CONTENTS

Details

This example creates a SAS data set in a Base SAS session by submitting the FEDSQL procedure, and then submitting FedSQL statements. The PROC CONTENTS output lists the contents of the SAS data set.

Program

```
libname myfiles base 'C:\My Documents';

proc fedsql;


create table myfiles.customer
   ( id double primary key,
     name char(16),
     address char(64),
     city char(16),
     state char(2),
     country char(16),
     homephone char(16),
     workphone char(16),
     cellphone char(16),
     initorder double having format date9. label 'Initial Order'
   );

quit;

proc contents data=myfiles.customer;
run;
```

Program Description

Assign a library reference to the SAS data set to be created. The LIBNAME statement assigns the libref MyFiles, specifies the BASE engine, and specifies the physical location for the SAS data set.

```
libname myfiles base 'C:\My Documents';
```

Execute the PROC FEDSQL statement. The PROC FEDSQL statement generates a connection string to the data source from the libref attributes and sets up an environment to submit FedSQL statements.

```
proc fedsql;
```

Enter the FedSQL statements. The FedSQL statements create the SAS data set named MyFiles.Customer. Note that the two-level name in the FedSQL CREATE TABLE statement specifies the catalog identifier MyFiles, which is the assigned libref.

```
create table myfiles.customer
   ( id double primary key,
     name char(16),
     address char(64),
     city char(16),
     state char(2),
     country char(16),
     homephone char(16),
     workphone char(16),
     cellphone char(16),
     initorder double having format date9. label 'Initial Order'
   );
```

Stop the procedure. The QUIT statement stops the procedure.

```
quit;
```

List the contents of the SAS data set. The CONTENTS procedure describes the contents of the SAS data set.

```
proc contents data=myfiles.customer;
run;
```

Output: Creating a SAS Data Set

Output 8.1 *PROC CONTENTS Output of MyFiles.Customer*

The SAS System

The CONTENTS Procedure

Data Set Name	MYFILES.CUSTOMER	Observations	0
Member Type	DATA	Variables	10
Engine	BASE	Indexes	1
Created	09/26/2012 15:02:10	Integrity Constraints	1
Last Modified	09/26/2012 15:02:10	Observation Length	184
Protection		Deleted Observations	0
Data Set Type		Compressed	NO
Label		Sorted	NO
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	16384
Number of Data Set Pages	2
First Data Page	1
Max Obs per Page	88
Obs in First Data Page	0
Index File Page Size	4096
Number of Index File Pages	2
Number of Data Set Repairs	0
Filename	C:\My Documents\customer.sas7bdat
Release Created	5.1TK
Host Created	W32_7

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
3	ADDRESS	Char	64		
9	CELLPHONE	Char	16		
4	CITY	Char	16		
6	COUNTRY	Char	16		
7	HOMEPHONE	Char	16		
1	ID	Num	8		
10	INITORDER	Num	8	DATE9.	Initial Order
2	NAME	Char	16		
5	STATE	Char	2		
8	WORKPHONE	Char	16		

Alphabetic List of Integrity Constraints

#	Integrity Constraint	Type	Variables
1	_PK0001_	Primary Key	ID

Alphabetic List of Indexes and Attributes

#	Index	Unique Option	Owned by IC	# of Unique Values
1	ID	YES	YES	0

Example 2: Creating a SAS Data Set from Existing Tables

Features: PROC FEDSQL statement
QUIT statement

Other features: LIBNAME statement
FedSQL language statements

Details

This example creates a table from existing tables by using PROC FEDSQL and the CREATE TABLE statement with the AS query expression syntax. The query expression selects rows from existing tables to create the new table.

Program

```
libname mybase base 'C:\base';

proc fedsql;

   create table mybase.outtable as
      select product.prodid, product.product, customer.name,
         sales.totals, sales.country
      from mybase.product, mybase.sales, mybase.customer
      where product.prodid = sales.prodid and
         customer.custid = sales.custid;

   select * from mybase.outtable;

quit;
```

Program Description

Assign a library reference. The LIBNAME statement assigns the libref MyBase, specifies the BASE engine, and specifies the physical location of the SAS data sets that will be used as input tables. The output table will be created in the same location.

```
libname mybase base 'C:\base';
```

Execute the PROC FEDSQL statement. The PROC FEDSQL statement connects to the data source from the libref attributes.

```
proc fedsql;
```

Create the new table. The CREATE TABLE statement creates a new SAS data set from three existing SAS data sets by using a query expression to select rows from the existing data sets. The SELECT statement retrieves the qualified columns and rows from the existing data sets to create the new SAS data set.

```
create table mybase.outtable as
   select product.prodid, product.product, customer.name,
      sales.totals, sales.country
   from mybase.product, mybase.sales, mybase.customer
   where product.prodid = sales.prodid and
      customer.custid = sales.custid;
```

Retrieve data in the SAS data set. The SELECT statement retrieves the columns and rows from the SAS data set named OutTable.

```
select * from mybase.outtable;
```

Stop the procedure.

```
quit;
```

Output: Creating a Table from Existing Tables

Output 8.2 *SELECT Statement for MyBase.OutTable*

The SAS System

PRODID	PRODUCT	NAME	TOTALS	COUNTRY
3234	Rice	Peter Frank	189400	United States
3422	Oat	Jim Stewart	2789654	United States
1424	Corn	Janet Chien	555789	Japan
3421	Wheat	Qing Ziao	781183	Japan
3975	Barley	Humberto Sertu	899453	Argentina

Example 3: Loading and Joining Tables in CAS

Features: CAS system options
CAS statement
CASLIB statement
CAS LIBNAME statement
DATA step
CASUTIL procedure
PROC FEDSQL statement
SESSREF= procedure option

Details

This example loads three files containing comma-delimited data into CAS and then executes the PROC FEDSQL CREATE TABLE statement with the AS query expression syntax to create a new table. A query expression selects columns and rows from the existing tables to create the new table. The input files are named Supplier, Nation, and Customer. The FedSQL output table is named NewTable.

Note: When formatting your FEDSQL requests, be aware that leading spaces before statements and clauses are important. Do not begin statements and clauses flush to the left margin. If you put a line break in a quoted string, always follow the line break with at least one blank.

Program

```
options cashost="cloud.example.com" casport=5570;

cas mysess;

caslib casdata path='/r/ge.unx.company.com/vol/vol210/u21/myID/hold';

libname mycas host="cloud.example.com" port=5570 sessref=mysess caslib=casdata;

data mycas.supplier;
  infile "u:\hold\supplier.tbl" delimiter='|';
  length S_SUPPKEY 8. S_NAME VARCHAR(25) S_ADDRESS VARCHAR(40) S_NATIONKEY 8.
S_PHONE VARCHAR(15) S_ACCTBAL 8. S_COMMENT VARCHAR(101);
  input  S_SUPPKEY S_NAME S_ADDRESS S_NATIONKEY S_PHONE S_ACCTBAL S_COMMENT;
run;

data mycas.nation;
  infile "u:\hold\nation.tbl" delimiter='|';
  length N_NATIONKEY 8. N_NAME VARCHAR(25) N_REGIONKEY 8. N_COMMENT VARCHAR(152);
  input  N_NATIONKEY N_NAME N_REGIONKEY N_COMMENT;
run;

data mycas.customer;
  infile "u:\hold\customer.tbl" delimiter='|';
  length C_CUSTKEY 8. C_NAME VARCHAR(25) C_ADDRESS VARCHAR(40) C_NATIONKEY 8.
 C_PHONE VARCHAR(15) C_ACCTBAL 8. C_MKTSEGMENT VARCHAR(10) C_COMMENT VARCHAR(117);
  input  C_CUSTKEY C_NAME C_ADDRESS C_NATIONKEY C_PHONE C_ACCTBAL C_MKTSEGMENT
  C_COMMENT;
run;

proc casutil;
  list tables incaslib="casdata";
run;

proc fedsql sessref=mysess;

  create table newtable {options replace=true} as
  select
    s_name, s_acctbal, n_name, sum_c_acctbal
  from
    supplier,
    nation,
    (select c_nationkey, sum(c_acctbal) as sum_c_acctbal from customer group by
 c_nationkey) C
   where
    s_nationkey = n_nationkey and
    s_nationkey = c_nationkey
  ;

select * from newtable;

quit;
```

Program Description

Invoke the SAS Cloud Analytic Services server. The CASHOST= and CASPORT= system options specify the name and port number of the CAS Server.

```
options cashost="cloud.example.com" casport=5570;
```

Establish a session on the CAS server. The CAS statement specifies to create a CAS session named MySess.

```
cas mysess;
```

Assign a CAS library that points to your input files. The CASLIB statement assigns the caslib CASDATA to the location specified in the PATH= parameter. The path specification must use an absolute pathname.

```
caslib casdata path='/r/ge.unx.company.com/vol/vol210/u21/myID/hold';
```

Assign a libref that specifies the CAS engine and connection parameters for the CAS server. The LIBNAME statement assigns the libref MyCas and associates it with the CAS engine and the CAS server. The LIBNAME statement also specifies the caslib.

```
libname mycas host="cloud.example.com" port=5570 sessref=mysess caslib=casdata;
```

Load the first file into CAS. The DATA step uses the INFILE= statement to read the contents of the file named Supplier into a table named Supplier. The libref MyCas specifies to create the output table as a CAS table. Table MyCas.Supplier has columns S_SUPPKEY, S_NAME, S_ADDRESS, S_NATIONKEY, S_PHONE, S_ACCTBAL, and S_COMMENT. Note that the INFILE specification is relative to the path specified in the CASLIB statement.

```
data mycas.supplier;
  infile "u:\hold\supplier.tbl" delimiter='|';
  length S_SUPPKEY 8. S_NAME VARCHAR(25) S_ADDRESS VARCHAR(40) S_NATIONKEY 8.
S_PHONE VARCHAR(15) S_ACCTBAL 8. S_COMMENT VARCHAR(101);
  input  S_SUPPKEY S_NAME S_ADDRESS S_NATIONKEY S_PHONE S_ACCTBAL S_COMMENT;
run;
```

Load the second file into CAS. The DATA step uses the INFILE= statement to read the contents of the file named Nation into a table named Nation. The libref MyCas specifies to create the output table as a CAS table. Table MyCas.Nation has columns N_NATIONKEY, N_NAME, N_REGIONKEY, and N_COMMENT.

```
data mycas.nation;
  infile "u:\hold\nation.tbl" delimiter='|';
  length N_NATIONKEY 8. N_NAME VARCHAR(25) N_REGIONKEY 8. N_COMMENT VARCHAR(152);
  input  N_NATIONKEY N_NAME N_REGIONKEY N_COMMENT;
run;
```

Load the third file into CAS. The DATA step uses the INFILE= statement to read the contents of the file named Customer into a table named Customer. The libref MyCas specifies to create the output table as a CAS table. Table MyCas.Customer has columns C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE, C_ACCTBAL, C_MKTSEGMENT, and S_COMMENT.

```
data mycas.customer;
  infile "u:\hold\customer.tbl" delimiter='|';
  length C_CUSTKEY 8. C_NAME VARCHAR(25) C_ADDRESS VARCHAR(40) C_NATIONKEY 8.
 C_PHONE VARCHAR(15) C_ACCTBAL 8. C_MKTSEGMENT VARCHAR(10) C_COMMENT VARCHAR(117);
  input  C_CUSTKEY C_NAME C_ADDRESS C_NATIONKEY C_PHONE C_ACCTBAL C_MKTSEGMENT
  C_COMMENT;
run;
```

Verify that the files were created. Submit the CASUTIL procedure to list the tables that are available in the active library.

```
proc casutil;
  list tables incaslib="casdata";
run;
```

Execute the PROC FEDSQL statement with the SESSREF= procedure option. The SESSREF= procedure option specifies the value MySess to connect to the CAS session.

```
proc fedsql sessref=mysess;
```

Submit FedSQL statements. The CREATE TABLE statement specifies to create a table NewTable using columns from the Supplier, Nation, and Customer tables. The SELECT statement retrieves the columns S_NAME, S_ACCTBAL, N_NAME from the tables and creates a new column SUM_C_ACCTBAL by issuing a subquery. The tables are joined based on the values in the S_NATIONKEY, N_NATIONKEY, and C_NATIONKEY columns. FedSQL combines the data and returns the results in a CAS table.

```
 create table newtable {options replace=true} as
 select
   s_name, s_acctbal, n_name, sum_c_acctbal
 from
   supplier,
   nation,
   (select c_nationkey, sum(c_acctbal) as sum_c_acctbal from customer group by
c_nationkey) C
  where
   s_nationkey = n_nationkey and
   s_nationkey = c_nationkey
 ;
```

Display the contents of table NewTable. The SELECT statement specifies to print CAS table NewTable. NewTable is an in-memory table. To persist or promote it, you must use PROC CASUTIL.

```
select * from newtable;
```

Stop the procedure.

```
quit;
```

Output: Joining CAS Tables

Output 8.3 *Output of the CASUTIL Procedure LIST Statement*

The CASUTIL Procedure

Table Information for Caslib CASDATA

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
SUPPLIER	10000	7	utf-8	08Jul2016:13:57:25	08Jul2016:13:57:25	No	No	No	No
NATION	25	4	utf-8	08Jul2016:13:57:43	08Jul2016:13:57:43	No	No	No	No
CUSTOMER	150000	8	utf-8	08Jul2016:13:58:14	08Jul2016:13:58:14	No	No	No	No

Output 8.4 *A Portion of the SELECT Results for Table NewTable*

S_NAME	S_ACCTBAL	N_NAME	SUM_C_ACCTBAL
Supplier#000000014	9189.820000	MOROCCO	26625512
Supplier#000000153	850.550000	INDONESIA	27930482
Supplier#000000292	9598.620000	VIETNAM	27081998
Supplier#000000431	9477.340000	CANADA	27025344
Supplier#000000570	922.720000	PERU	26556292
Supplier#000000709	8638.350000	SAUDI ARABIA	26455693
Supplier#000000848	4404.290000	CANADA	27025344
Supplier#000000987	-40.300000	CHINA	26740212
Supplier#000001126	7243.050000	JAPAN	26898469
Supplier#000001265	3986.630000	IRAQ	26919597
Supplier#000001404	5223.720000	ALGERIA	26322970
Supplier#000001543	2107.210000	EGYPT	27100354
Supplier#000001682	9413.980000	CANADA	27025344
Supplier#000001821	7431.290000	VIETNAM	27081998
Supplier#000001960	443.600000	SAUDI ARABIA	26455693
Supplier#000002099	7043.940000	ALGERIA	26322970
Supplier#000002238	5244.930000	JORDAN	26891366
Supplier#000002377	7455.660000	MOZAMBIQUE	27022906
Supplier#000002516	2819.790000	EGYPT	27100354
Supplier#000002655	4856.740000	VIETNAM	27081998
Supplier#000002794	4271.280000	KENYA	27406567
Supplier#000002933	3557.950000	BRAZIL	26821676
Supplier#000003072	7279.730000	INDIA	27293627
Supplier#000003211	8315.050000	UNITED STATES	27316299

Example 4: Obtain Query Details with the _METHOD= Option

Features: PROC FEDSQL statement
_METHOD procedure option
SESSREF= procedure option

Details

This example executes the code from Example 3 with the _METHOD procedure option. The _METHOD option specifies to print a textual description of the query plan for the specified FedSQL query. This example assumes that you are executing this query in the same CAS session as Example 3 and have access to the tables that were loaded for Example 3.

```
proc fedsql sessref=mysess _method=true;
  create table fedsql_out {options replace=true} as
  select
    s_name, s_acctbal, n_name, sum_c_acctbal
  from
    SUPPLIER,
    NATION,
    (select c_nationkey, sum(c_acctbal) as sum_c_acctbal from CUSTOMER group by
```

```
c_nationkey) C
  where
   s_nationkey = n_nationkey and
   s_nationkey = c_nationkey
 ;
```

Output: Obtain Query Details with the _METHOD= Option

Output 8.5 *Log Information Written by the _METHOD= Option*

```
Methods for full query plan
 ----------------------------
 Number of Sorts Performed is : 1
 Number of Joins Performed is : 2
         HashJoin (INNER)
           SubqueryScan
             Agg
               Sort
                 SeqScan from CASUSER(userid).CUSTOMER
             HashJoin (INNER)
               SeqScan from CASUSER(userid).SUPPLIER
                 SeqScan from CASUSER(userid).NATION

 Methods for stage 1
 --------------------
         Agg
             SeqScan with _pushed_ order by from CASUSER(userid).CUSTOMER

         Stage query: create table "CASUSER(userid)"."__fedsql_1__" {options
replace=true} as select "T1"."C_NATIONKEY", SUM
 ("T1"."C_ACCTBAL")  as "SUM_C_ACCTBAL" from "CASUSER(userid)"."CUSTOMER" T1
group by "T1"."C_NATIONKEY"

 Methods for stage 3
 --------------------
         HashJoin (INNER)
           SeqScan from CASUSER(userid).SUPPLIER
             SeqScan from CASUSER(userid).NATION

         Stage query: create table "CASUSER(userid)"."__fedsql_3__" {options
replace=true} as select "T2"."N_NAME",
 "T2"."N_NATIONKEY", "T1"."S_NAME", "T1"."S_ACCTBAL", "T1"."S_NATIONKEY" from
"CASUSER(userid)"."SUPPLIER" T1 _hash_ inner join
 "CASUSER(userid)"."NATION" {options REPL=YES } T2 on
("T1"."S_NATIONKEY"="T2"."N_NATIONKEY")

 Methods for stage 4
 --------------------
         HashJoin (INNER)
           SeqScan from CASUSER(userid).__fedsql_3__
             SeqScan from CASUSER(userid).__fedsql_1__

         Stage query: create table "CASUSER(userid)"."FEDSQL_OUT" {options
replace=true} as select "T2"."S_NAME", "T2"."S_ACCTBAL",
 "T2"."N_NAME", "T1"."SUM_C_ACCTBAL" from "CASUSER(userid)"."__fedsql_3__" T2
_hash_ inner join "CASUSER(userid)"."__fedsql_1__"
 {options REPL=YES } T1 on  ("T1"."C_NATIONKEY"="T2"."S_NATIONKEY")
```

Chapter 9

FORMAT Procedure

Overview: FORMAT Procedure

What Does the FORMAT Procedure Do?

The FORMAT procedure enables you to define your own informats and formats for variables. In addition, you can perform these actions:

- print the parts of a catalog that contain informats or formats
- store descriptions of informats or formats in a SAS data set
- use a SAS data set to create informats or formats.

Format Catalog Encodings in SAS Viya

SAS Viya supports only the UTF-8 encoding.

For more information about the encodings of format catalogs, see *Migrating Data to UTF-8 for SAS Viya* and *SAS Viya FAQ for Processing UTF-8 Data*.

Note: Moving catalogs between previous versions of SAS and CAS might have some risk. SAS recommends that you use CNTLOUT data sets to reduce this risk.

What Are Formats and Informats?

Informats determine how raw data values are read and stored. Formats determine how variable values are printed. For simplicity, this section uses the terminology the informat converts and the format prints.

Informats and formats tell SAS the data's type (character or numeric) and form (such as how many bytes it occupies; decimal placement for numbers; how to handle leading, trailing, or embedded blanks and zeros; and so on). SAS provides informats and formats for reading and writing variables. For a thorough description of informats and formats that SAS provides, see *SAS Viya Formats and Informats: Reference*.

With informats, you can do the following:

- Convert a number to a character string (for example, convert 1 to **YES**).
- Convert a character string to a different character string (for example, convert **'YES'** to **'OUI'**).
- Convert a character string to a number (for example, convert **YES** to 1).
- Convert a number to another number (for example, convert 0–9 to 1, 10–100 to 2, and so on).

Note: User-defined informats read-only character data. They can convert character values into real numeric values, but they cannot convert real numbers into characters.

With formats, you can do the following:

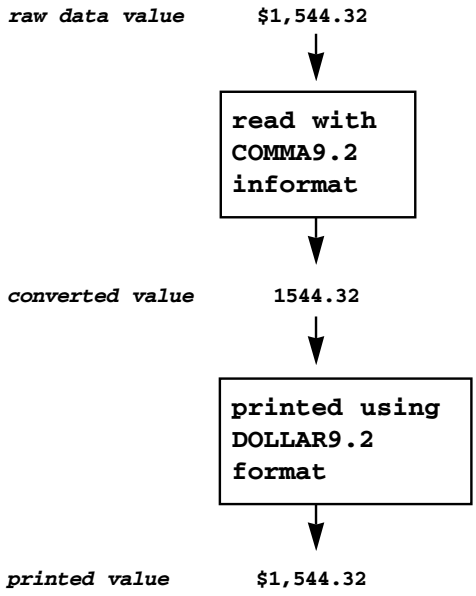
- Print numeric values as character values (for example, print 1 as **MALE** and 2 as **FEMALE**).

- Print one character string as a different character string (for example, print **YES** as **OUI**).
- Print numeric values using a template (for example, print 9458763450 as **945-876-3450**).

How Are Formats and Informats Associated with a Variable?

The following figure summarizes what occurs when you associate an informat and format with a variable. The COMMA*w.d* informat and the DOLLAR*w.d* format are provided by SAS.

Figure 9.1 *Associating an Informat and a Format with a Variable*



In the figure, SAS reads the raw data value that contains the dollar sign and comma. The COMMA9.2 informat ignores the dollar sign and comma and converts the value to 1544.32. The DOLLAR9.2 format prints the value, adding the dollar sign and comma. For more information about associating informats and formats with variables, see "Associating Informats and Formats with Variables " on page 149.

Concepts: FORMAT Procedure

Associating Informats and Formats with Variables

Methods of Associating Informats and Formats with Variables

The following table summarizes the different methods for associating informats and formats with variables.

Table 9.1 *Associating Informats and Formats with Variables*

Step	Informats	Formats
In a DATA step	Use the ATTRIB or INFORMAT statement to permanently associate an informat with a variable. Use the INPUT function or INPUT statement to associate the informat with the variable only for the duration of the DATA step.	Use the ATTRIB or FORMAT statement to permanently associate a format with a variable. Use the PUT function or PUT statement to associate the format with the variable only for the duration of the DATA step.
In a PROC step	The ATTRIB and INFORMAT statements are valid in SAS procedures. However, in SAS software, typically you do not assign informats in PROC steps because the data has already been read into SAS variables.	Use the ATTRIB statement or the FORMAT statement to associate formats with variables. If you use either statement in a procedure that produces an output data set, then the format is permanently associated with the variable in the output data set. If you use either statement in a procedure that does not produce an output data set or modify an existing data set, the statement associates the format with the variable only for the duration of the PROC step.

Differences between the FORMAT Statement and PROC FORMAT

Do not confuse the FORMAT statement with the FORMAT procedure. The FORMAT and INFORMAT statements associate an existing format or informat (either standard SAS or user-defined) with one or more variables. PROC FORMAT creates user-defined formats or informats.

Assigning Formats and Informats to a Variable

Assigning your own format or informat to a variable is a two-step process:

1. creating the format or informat with the FORMAT procedure
2. assigning the format or informat with the ATTRIB, FORMAT,or INFORMAT statements, or the INPUT or PUT functions

For complete documentation on the ATTRIB, INFORMAT, and FORMAT statements, see *SAS Viya Statements: Reference*. For complete documentation on the INPUT and PUT functions, see *SAS Viya Functions and CALL Routines: Reference*.

Storing Informats and Formats

Format Catalogs

PROC FORMAT stores user-defined informats and formats as entries in SAS catalogs.[1] You use the LIBRARY= option in the PROC FORMAT statement to specify the catalog. If you omit the LIBRARY= option, then formats and informats are stored in the Work.Formats catalog. If you specify LIBRARY=*libref* but do not specify a catalog name, then formats and informats are stored in the *libref*.FORMATS catalog. Note that this use of a one-level name differs from the use of a one-level name elsewhere in SAS.

[1] Catalogs are a type of SAS file and reside in a SAS library. If you are unfamiliar with the types of SAS files or the SAS library structure, then see the section on SAS files in *SAS Language Reference: Concepts*.

With the LIBRARY= option, a one-level name indicates a library; elsewhere in SAS, a one-level name indicates a file in the WORK library.

The name of the catalog entry is the name of the format or informat. The entry types are as follows:

- FORMAT for numeric formats
- FORMATC for character formats
- INFMT for numeric informats
- INFMTC for character informats

Temporary Informats and Formats

Informats and formats are temporary when they are stored in a catalog in the WORK library. If you omit the LIBRARY= option, then PROC FORMAT stores the informats and formats in the temporary catalog Work.Formats. You can retrieve temporary informats and formats only in the same SAS session or job in which they are created. To retrieve a temporary format or informat, simply include the name of the format or informat in the appropriate SAS statement. SAS automatically looks for the format or informat in the Work.Formats catalog.

Permanent Informats and Formats

If you want to use a format or informat that is created in one SAS job or session in a subsequent job or session, then you must permanently store the format or informat in a SAS catalog.

You permanently store informats and formats by using the LIBRARY= option in the PROC FORMAT statement. See the discussion of the LIBRARY= option in the PROC FORMAT Statement on page 155.

Accessing Permanent Informats and Formats

After you have permanently stored an informat or format, you can use it in later SAS sessions or jobs. If you associate permanent informats or formats with variables in a later SAS session or job, then SAS must be able to access the informats and formats. Thus, you must use a LIBNAME statement to assign a libref to the library that stores the catalog that stores the informats or formats.

SAS uses one of two methods when searching for user-defined formats and informats:

- By default, SAS always searches a library that is referenced by the Library libref for a FORMATS catalog. If you have only one format catalog, then do the following:
  1. Assign the Library libref to a SAS library in the SAS session in which you are running the PROC FORMAT step.
  2. Specify the option **`library=library`** in the PROC FORMAT statement. PROC FORMAT stores the informats and formats that are defined in that step in the Library.Formats catalog.
  3. In the SAS program that uses your user-defined formats and informats, include a LIBNAME statement to assign the Library libref to the library that contains the permanent format catalog.
- If you have more than one format catalog, or if the format catalog is named something other than Formats, then do the following:
  1. Assign a libref to a SAS library in the SAS session in which you are running the PROC FORMAT step.

2. Specify the option **library=*libref*** or **library=*libref.catalog*** in the PROC FORMAT step, where *libref* is the libref that you assigned in step 1.
3. In the SAS program that uses your user-defined formats and informats, use the FMTSEARCH= option in an OPTIONS statement, and include *libref* or *libref.catalog* in the list of format catalogs.

The syntax for specifying a list of format catalogs to search is

OPTIONS FMTSEARCH=(*catalog-specification-1*<*catalog-specification-2* … >);

Each *catalog-specification* can be *libref* or *libref.catalog*. If only *libref* is specified, then SAS assumes that the catalog name is Formats.

When searching for a format or informat, SAS always searches in Work.Formats first, and then Library.Formats, unless one of them appears in the FMTSEARCH= list. SAS searches the catalogs in the FMTSEARCH= list in the order in which they are listed until the format or informat is found.

For more information, see "FMTSEARCH= System Option" in *SAS Viya System Options: Reference*.

Missing Informats and Formats

If you reference an informat or format that SAS cannot find, then you receive an error message and processing stops unless the SAS system option NOFMTERR is in effect. When NOFMTERR is in effect, SAS uses the *w.* or $*w.* default format to print values for variables with formats that it cannot find. For example, to use NOFMTERR, use this OPTIONS statement:

```
options nofmterr;
```

For more information, see "FMTERR System Option" in *SAS Viya System Options: Reference*.

If SAS encounters a missing variable to format using a user-defined format and the MISSING= system option defines a character to be printed for missing values, the missing value is determined as follows:

- If the user-defined format or informat has a value-range-set for missing values, the missing value is defined by the user-defined format.
- If the user-defined format does not have a value-range-set defined for missing values, the missing value is defined by the MISSING= system option. The default value for the MISSING= system option is . (period).

Printing Informats and Formats

The output that is provided when you use the FMTLIB option in the PROC FORMAT statement is intended to present a brief view of the informat and format values.

Instead of using the FMTLIB option, you can use the CNTLOUT= option to create an output data set that stores information about informats and formats. You can then use PROC PRINT to print the data set. In this case, labels are not truncated.

Note: You can use data set options to keep or drop references to additional variables that were added by using the CNTLOUT= option.

Using Formats in a CAS Session

PROC FORMAT supports creating format libraries in catalogs in a SAS client session and loading format libraries to a SAS Cloud Analytics Services (CAS) session. For more information, see "CASFMTLIB='*name*'" on page 156.

When you use SAS Studio, the SAS client session runs on only the workspace server. The workspace server is on a single server that can be in a cloud environment or a local environment. The CAS session uses multiple worker nodes on several machines.

The SAS client session and the CAS session can interact through the session reference that you establish with the SESSREF option. When you use a SAS language element that can take advantage of processing on CAS, the session reference identifies where that processing should occur. If you do not specify a session reference, then processing occurs on the client session. If the language element is not supported in CAS, then processing occurs on the client session.

If you specify a CAS session, and specify the CASFMTLIB option, then PROC FORMAT loads the format libraries to the CAS session. Otherwise, PROC FORMAT stores the format libraries where the SAS client session is running.

When you log on to SAS Studio, it creates a primary session for you that has a default name of STUDIO. You can use this default session, or you can specify a different session with the CAS statement. However, specifying **`cas mysess`** does not override the primary session, and everything is written to the STUDIO session. To load your formats to your current CAS session, you have to specify the name of your current session with the SESSREF option. For example, to load your formats to a session named Mysess, specify **`sessref=mysess`**.

For more information, see the following documentation about CAS:

- *SAS Cloud Analytic Services: Fundamentals*
- *SAS Cloud Analytic Services: Language Reference*
- *SAS Cloud Analytic Services: Accessing and Manipulating Data*

A Binary Search Determines the User-Defined Format or Informat for a Value

SAS uses a binary search to determine the correct user-defined format or informat to use for a value. To compare, using IF-THEN/ELSE statements is essentially a sequential search for a value.

Here are some user-defined format values that could be written using PROC FORMAT:

```
1='Yes'
2='No'
3='Possibly'
```

SAS makes a single comparison to format a value.

Using these IF-THEN/ELSE statements, SAS makes a single comparison to format a value as well.:

```
if x=1 then label='Yes';
else if x=2 then label='No';
else if x=3 then label='Possibly';
```

A binary search is more efficient when the number of comparisons to make increases. The greater the number of comparisons, the more efficient is the binary search.

Syntax: FORMAT Procedure

Restrictions: You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

When the CASFMTLIB option is specified, the SELECT and EXCLUDE statements ignore format libraries in SAS Cloud Analytics Services (CAS) sessions and refer only to catalogs.

Formats that are not enabled for threaded-kernel processing are not written to CAS.

Informats cannot be written to a CAS session. Informats included in your SAS code are ignored.

Formats that use functions-as-labels or formats-as-labels cannot be written to CAS.

Tip: You can also use appropriate global statements with this procedure.

See: "Using Formats in a CAS Session" on page 153

SAS Viya supports only the UTF-8 encoding. For information about the encoding of your data sets in SAS Viya, see *Migrating Data to UTF-8 for SAS Viya* and *SAS Viya FAQ for Processing UTF-8 Data*.

PROC FORMAT *<option(s)>*;
  EXCLUDE *entry(s)*;
  INVALUE <$>*name* <(*informat-option(s)*)> <*value-range-set(s)*>;
  PICTURE *name* <(*format-option(s)*)>
    <*value-range-set-1* <(*picture-1-option(s)*)>
    <*value-range-set-2* <(*picture-2-option(s)*)>> …>;
  SELECT *entry(s)*;
  VALUE <$>*name* <(*format-option(s)*)> <*value-range-set(s)*>;

Statement	Task	Example
PROC FORMAT	Define formats and informats for variables	Ex. 3, Ex. 8, Ex. 9
EXCLUDE	Exclude catalog entries from processing by the FMTLIB and CNTLOUT= options	
INVALUE	Create an informat for reading and converting raw data values	
PICTURE	Create a template for printing numbers	Ex. 3, Ex. 5, Ex. 10
SELECT	Select catalog entries for processing by the FMTLIB and CNTLOUT= options	
VALUE	Create a format that specifies character strings to use to print variable values	Ex. 6

PROC FORMAT Statement

Creates user-specified formats and informats for variables.

Tips: You can use data set options with the CNTLIN= and CNTLOUT= data set options.

Moving catalogs between previous versions of SAS and CAS might have some risk. SAS recommends that you use CNTLOUT data sets to reduce this risk.

See: For information about formats that support threaded processing, see *SAS Viya Formats and Informats: Reference*

Examples: "Example 3: Creating a Picture Format" on page 204

"Example 8: Creating a Format from a Data Set" on page 215

"Example 9: Retrieving a Permanent Format" on page 218

Syntax

PROC FORMAT *<option(s)>*;

Summary of Optional Arguments

CASFMTLIB='*name*'
: adds a format library to a CAS session.

CNTLIN=*input-control-SAS-data-set*
: specifies a SAS data set from which PROC FORMAT builds informats or formats.

CNTLOUT=*output-control-SAS-data-set*
: creates a SAS data set that stores information about informats or formats that are contained in the catalog specified in the LIBRARY= option.

FMTLIB
: prints information about informats or formats in the catalog that is specified in the LIBRARY= option.

LIBRARY=*libref<.catalog>*
: specifies a SAS library or catalog that contains the informats or formats that you are creating in the PROC FORMAT step.

LOCALE
: specifies to create a format catalog that corresponds to the current SAS locale.

MAXLABLEN=*number-of-characters*
: specifies the number of characters of the informatted or formatted value that appear in PROC FORMAT output.

MAXSELEN=*number-of-characters*
: specifies the number of characters of the start and end values that appear in the PROC FORMAT output.

NOREPLACE
: prevents a new informat or format from replacing an existing one of the same name.

PAGE
: prints information about each format and informat in the catalog.

Optional Arguments

CASFMTLIB='*name*'
adds a format library to a CAS session.

You can specify the CASFMTLIB option only in an active SAS Cloud Analytics Services (CAS) session. PROC FORMAT connects to the CAS session and loads a format library. If the format library already exists in the CAS session, then SAS updates it. SAS also appends the format library to the search list for any subsequent referencing by procedures that are operating in CAS in that session. That is, if a format library already exists and you create a new library with CASFMTLIB, then the new library is appended to the search order. The library name should be a one-level name that does not contain any slashes.

You can specify additional CAS sessions with the SESSREF= option. For information about the SESSREF= option and other CAS language elements, see *SAS Cloud Analytic Services: Language Reference*.

SAS formats are available in the local SAS client regardless of whether you add them to a format library in CAS. When the CASFMTLIB option is specified, the EXCLUDE and SELECT statements are applied to the local SAS session format catalogs, not to the CAS session format library. Informats cannot be loaded into a CAS session. If you specify an INVALUE statement with CASFMTLIB, then a note is written to the log and nothing is written to the CAS format library.

Notes The specified value of the CASFMTLIB option must be enclosed in single or double quotation marks.

The library name that you specify is case-sensitive. It has to be a valid SAS name, and cannot contain blank spaces.

Formats defined in the VALUE or PICTURE statements are also written to the format catalog that is specified by the LIBRARY= option. If you do not specify a library, then SAS uses the WORK.FORMATS library.

Tip You can use the CAS action **`addFmtLib=fmtsearch`** to control the order in which SAS searches for format libraries. For more information, see "Manage User-Defined Formats with CAS Actions" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*

See "Using Formats in a CAS Session" on page 153

"Example 1: Create a Format Library in a CAS Session" on page 199

"Using Formats in a CAS Session" on page 153

CNTLIN=*input-control-SAS-data-set*
specifies a SAS data set from which PROC FORMAT builds informats or formats.

CNTLIN= builds formats and informats without using a VALUE, PICTURE, or INVALUE statement. If you specify a one-level name, then the procedure searches only the default library (either the WORK library or USER library) for the data set, regardless of whether you specify the LIBRARY= option.

Note LIBRARY= can point to either a library or a catalog. If only a libref is specified, a catalog name of FORMATS is assumed.

Tip A common source for an input control data set is the output from the CNTLOUT= option of another PROC FORMAT step.

Example "Example 8: Creating a Format from a Data Set" on page 215

CNTLOUT=*output-control-SAS-data-set*

creates a SAS data set that stores information about informats or formats that are contained in the catalog specified in the LIBRARY= option. If you are creating an informat or format in the same step that the CNTLOUT= option appears, then the informat or format that you are creating is included in the CNTLOUT= data set.

If you specify a one-level name, then the procedure stores the data set in the default library (either the WORK library or the USER library), regardless of whether you specify the LIBRARY= option.

Note LIBRARY= can point to either library or a catalog. If only a libref is specified, SAS uses the catalog name FORMATS.

Tip You can use an output control data set as an input control data set in subsequent PROC FORMAT steps.

See "Output Control Data Set" on page 193

SAS Viya supports only UTF-8 encoding. For information about the encoding of your format catalogs in SAS Viya, see *Migrating Data to UTF-8 for SAS Viya* and *SAS Viya FAQ for Processing UTF-8 Data*.

FMTLIB

prints information about informats or formats in the catalog that is specified in the LIBRARY= option. To get information about specific informats or formats, subset the catalog using the SELECT or EXCLUDE statement.

Note: FMTLIB is not supported for CASFMTLIB.

Interaction The PAGE option invokes FMTLIB.

Tips If your output from FMTLIB is not formatted correctly in the ODS LISTING destination, then try increasing the value of the LINESIZE= system option.

If you use the SELECT or EXCLUDE statement and omit the FMTLIB and CNTLOUT= options, then the procedure invokes the FMTLIB option and you receive FMTLIB option output.

LIBRARY=*libref<.catalog>*

specifies a SAS library or catalog that contains the informats or formats that you are creating in the PROC FORMAT step. The procedure stores these informats and formats in the catalog that you specify so that you can use them in subsequent SAS sessions or jobs.

Alias LIB=

Default If you omit the LIBRARY= option, then formats and informats are stored in the Work.Formats catalog. If you specify the LIBRARY= option but do not specify a name for *catalog*, then formats and informats are stored in the *libref*.FORMATS catalog.

Note LIBRARY= can point to either a library or a catalog. If only a libref is specified, then SAS uses the catalog name FORMATS.

Tips SAS automatically searches Library.Formats. You might want to define and use the LIBRARY libref for your format catalog.

You can control the order in which SAS searches for format catalogs with the FMTSEARCH= system option. For more information, see "FMTSEARCH= System Option" in *SAS Viya System Options: Reference*.

See "Storing Informats and Formats " on page 150

Example "Example 3: Creating a Picture Format" on page 204

LOCALE

specifies to create a format catalog that corresponds to the current SAS locale. The name of the catalog that SAS creates is the SAS library or catalog that is specified in the LIBRARY= option appended with the five-character POSIX locale value for the current SAS locale.

See For a list of POSIX locale values, see "LOCALE= Values for PAPERSIZE and DFLANG Options" in *SAS Viya National Language Support: Reference Guide*.

Example If the SAS locale is German_Germany, the POSIX locale value is de_DE. Using the following PROC FORMAT statement, SAS creates the catalog mylib.formats_de_DE to store formats and informats created by this procedure:

```
proc format locale lib=mylib.formats;
```

MAXLABLEN=*number-of-characters*

specifies the number of characters in the informatted or formatted value that you want to appear in the CNTLOUT= data set or in the output of the FMTLIB option. The FMTLIB option prints a maximum of 40 characters for the informatted or formatted value.

MAXSELEN=*number-of-characters*

specifies the number of characters in the start and end values that you want to appear in the CNTLOUT= data set or in the output of the FMTLIB option. The FMTLIB option prints a maximum of 16 characters for start and end values.

NOREPLACE

prevents a new informat or format from replacing an existing one of the same name. If you omit NOREPLACE, then the procedure warns you that the informat or format already exists and replaces it.

Note You can have a format and an informat of the same name.

PAGE

prints information about each format and informat in the catalog.

Interaction The PAGE option activates the FMTLIB option.

Tip In the ODS LISTING destination, the information about each format and informat appears on separate pages in the Results tab.

EXCLUDE Statement

Excludes entries from processing by the FMTLIB and CNTLOUT= options.

Restrictions: Only one EXCLUDE statement can appear in a PROC FORMAT step.

You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

When the CASFMTLIB option is specified, the EXCLUDE statement ignores format libraries in CAS sessions and refers only to catalogs in the SAS session.

Syntax

EXCLUDE *entry(s)*;

Required Argument

entry(s)

specifies one or more catalog entries to exclude from processing. Catalog entry names are the same as the name of the informat or format that they store. Because informats and formats can have the same name, and because character and numeric informats or formats can have the same name, you must use certain prefixes when specifying informats and formats in the EXCLUDE statement. Follow these rules when specifying entries in the EXCLUDE statement:

- Precede names of entries that contain character formats with a dollar sign ($).
- Precede names of entries that contain character informats with an at sign and a dollar sign (for example, @$*entry-name*).
- Precede names of entries that contain numeric informats with an at sign (@).
- Specify names of entries that contain numeric formats without a prefix.

Details

Shortcuts to Specifying Names

You can use the colon (:) and hyphen (-) wildcard characters to exclude entries. For example, the following EXCLUDE statement excludes all formats or informats that begin with the letter **a**.

```
exclude a:;
```

In addition, the following EXCLUDE statement excludes all formats or informats that occur alphabetically between **apple** and **pear**, inclusive:

```
exclude apple-pear;
```

FMTLIB Output

If you use the EXCLUDE statement without either FMTLIB or CNTLOUT= in the PROC FORMAT statement, then the procedure invokes the FMTLIB option and you receive FMTLIB option output.

INVALUE Statement

Creates an informat for reading and converting raw data values.

Restriction: Informats cannot be written to a CAS session. If you specify an INVALUE statement with CASFMTLIB, then a note is written to the log and nothing is written to the CAS format library.

See: *SAS Viya Formats and Informats: Reference* for documentation on informats supplied by SAS.

Syntax

INVALUE <$>*name* <(*informat-option(s)*)> <*value-range-set(s)*>;

Summary of Optional Arguments

DEFAULT=*length*
: specifies the default length of the informat.

FUZZ=*fuzz-factor*
: specifies a fuzz factor for matching values to a range.

JUST
: left-justifies all input strings before they are compared to ranges.

MAX=*length*
: specifies a maximum length for the informat.

MIN=*length*
: specifies a minimum length for the informat.

NOTSORTED
: stores values or ranges in the order in which you define them.

REGEXP
REGEXPE
: specifies that the preceding range is to be treated as a Perl regular expression.

UPCASE
: upper cases all input strings before they are compared to ranges.

Control the input template.

value-range-set(s)
: specifies the variable template for reading data.

Required Argument

name

names the informat that you are creating.

Restriction A user-defined informat name cannot be the same as an informat name that is supplied by SAS.

Requirement The name must be a valid SAS name. A numeric informat name can be up to 31 characters in length; a character informat name can be up to 30 characters in length and cannot end in a number. If you are creating a character informat, then use a dollar sign ($) as the first

character. Adding the dollar sign to the name is why a character informat is limited to 30 characters.

Interaction The maximum length of an informat name is controlled by the VALIDFMTNAME= system option. For more information, see *SAS Viya System Options: Reference*.

Tips Refer to the informat later by using the name followed by a period. However, do not use a period after the informat name in the INVALUE statement.

When SAS prints messages that refer to a user-written informat, the name is prefixed by an at sign (@). When the informat is stored, the at sign is prefixed to the name that you specify for the informat. The addition of the at sign to the name is why the name is limited to 31 or 30 characters. You need to use the at sign *only* when you are using the name in an EXCLUDE or SELECT statement; do not prefix the name with an at sign when you are associating the informat with a variable.

Optional Arguments

DEFAULT=*length*

specifies the default length of the informat. The value for DEFAULT= becomes the length of the informat if you do not give a specific length when you associate the informat with a variable.

Defaults For character informats, the length of the longest label

For numeric informats, 12 if you have numeric data to the left of the equal sign

For quoted strings, the length of the longest string

Range 1–32767

Tip As a best practice, if you specify an existing informat in a value-range set, always specify the DEFAULT= option.

FUZZ=*fuzz-factor*

specifies a fuzz factor for matching values to a range. If a number does not match or fall in a range exactly but comes within *fuzz-factor*, then the informat considers it a match. For example, the following INVALUE statement creates the LEVELS. informat, which uses a fuzz factor of .2:

```
invalue levels (fuzz=.2) 1='A'
                         2='B'
                         3='C';
```

FUZZ=.2 means that if a variable value falls within .2 of a value on either end of the range, then the informat uses the corresponding formatted value to store the variable value. So the LEVELS. informat saves the value 2.1 as **B**.

Tips Specify FUZZ=0 to save storage space when you use the INVALUE statement to create numeric informats.

Use a nonzero fuzz factor only with numbers that are very close but not an exact match. Ranges are stored internally in sorted order (unless the

NOTSORTED option is used), in order to perform a binary search. When a fuzz-factor is added to the end of one range and subtracted from the beginning of the next range, and the ranges overlap, the results can be unpredictable. A value is placed in the first range that is a match in the binary search. The exclusion operator is insufficient to override this binary search algorithm. As a best practice, when you use the exclusion operator, set FUZZ=0 or the NOTSORTED option.

A best practice is to use FUZZ=0 when you use the < exclusion operator with numeric informats.

JUST
left-justifies all input strings before they are compared to ranges.

MAX=*length*
specifies a maximum length for the informat. When you associate the informat with a variable, you cannot specify a width greater than the MAX= value.

Default 40

Range 1–32767

MIN=*length*
specifies a minimum length for the informat.

Default 1

Range 1–32767

NOTSORTED
stores values or ranges in the order in which you define them.

If you do not specify NOTSORTED, then values or ranges are stored in sorted order by default, and SAS uses a binary searching algorithm to locate the range that a particular value falls into. If you specify NOTSORTED, then SAS searches each range in the order in which you define them until a match is found.

Use NOTSORTED if one of the following is true:

- You know the likelihood of certain ranges occurring, and you want your informat to search those ranges first to save processing time.
- You want to preserve the order that you define ranges when you print a description of the informat using the FMTLIB option.
- You want to preserve the order that you define ranges when you use the ORDER=DATA option and the PRELOADFMT option to analyze class variables in PROC MEANS or PROC SUMMARY.

Do not use NOTSORTED if the distribution of values is uniform or unknown, or if the number of values is relatively small. The binary searching algorithm that SAS uses when NOTSORTED is not specified optimizes the performance of the search under these conditions.

SAS automatically sets the NOTSORTED option when you use the CPORT and CIMPORT procedures to transport informats or formats between operating environments with different standard collating sequences. This automatic setting of NOTSORTED can occur when you transport informats or formats between ASCII and EBCDIC operating environments. If this situation is undesirable, then do the following:

- Use the CNTLOUT= option in the PROC FORMAT statement to create an output control data set.
- Use the CPORT procedure to create a transport file for the control data set.
- Use the CIMPORT procedure in the target operating environment to import the transport file.
- In the target operating environment, use PROC FORMAT with the CNTLIN= option to build the formats and informats from the imported control data set.

REGEXP
REGEXPE

specifies that the preceding range is to be treated as a Perl regular expression. If you specify REGEXPE, the regular expression is expected to produce a modified result, as in using the substitute action.

During execution, all regular expressions are compiled and the input data is passed to the first expression to confirm a match. If there is a match, the corresponding label is used. If there is no match, the next range is compared. Ranges are not sorted and are processed in the order in which they were defined in the INVALUE statement or in the order in which they appear in the CNTLIN= data set.

The rules for regular expressions using the REGEXP option are the same as they are for the PRXPARSE function in the DATA step. The rules for the REGEXPE option are the same as they are for the PRXCHANGE function.

Interaction If you are using a CNTLIN= data set, the HLO variable contains P for REGEXP and E for REGEXPE.

UPCASE

converts all raw data values to uppercase before they are compared to the possible ranges. If you use UPCASE, then make sure the values or ranges that you specify are in uppercase.

value-range-set(s)

specifies raw data and values that the raw data will become. The *value-range-set(s)* can be one or more of the following:

value-or-range-1<, *value-or-range-2* …>=*informatted-value* | [*existing-informat*]

The informat converts the raw data to the values of *informatted-value* on the right side of the equal sign.

value-or-range

See "Specifying Values or Ranges" on page 191.

informatted-value

is the value that you want the raw data in *value-or-range* to become. Use one of the following forms for *informatted-value*:

'character-string'

is a character string up to 32,767 characters long. Typically, *character-string* becomes the value of a character variable when you use the informat to convert raw data. Use *character-string* for *informatted-value* only when you are creating a character informat. If you omit the single or double quotation marks around *character-string*, then the INVALUE statement assumes that the quotation marks are there.

For hexadecimal literals, you can use up to 32,767 typed characters, or up to 16,382 represented characters at two hexadecimal characters per represented character.

number
is a number that becomes the informatted value. Typically, *number* becomes the value of a numeric variable when you use the informat to convert raw data. Use *number* for *informatted-value* when you are creating a numeric informat. The maximum for *number* depends on the host operating environment.

ERROR
treats data values in the designated range as invalid data. SAS assigns a missing value to the variable, prints the data line in the SAS log, and issues a warning message.

SAME
prevents the informat from converting the raw data as any other value. For example, the following GROUP. informat converts values 01 through 20 and assigns the numbers 1 through 20 as the result. All other values are assigned a missing value.

```
invalue group 01-20= _same_
              other= .;
```

existing-informat
is an informat that is supplied by SAS or an existing user-defined informat. The informat that you are creating uses the existing informat to convert the raw data that match *value-or-range* on the left side of the equal sign. If you use an existing informat, then enclose the informat name in square brackets (for example, [date9.]) or with parentheses and vertical bars (for example, (|date9.|)). Do not enclose the name of the existing informat in single quotation marks.

Tip As a best practice, if you specify an existing informat in a value-range-set, always specify a default value by using the DEFAULT= option.

Examples

Example 1: Create a Character Informat for Raw Data Values

The $GENDER. character informat converts the raw data values **F** and **M** to character values **'1'** and **'2'**:

```
invalue $gender 'F'='1'
                'M'='2';
```

The dollar sign prefix indicates that the informat converts character data.

Example 2: Create Character and Numeric Values or a Range of Values

When you create numeric informats, you can specify character strings or numbers for *value-or-range*. For example, the TRIAL. informat converts any character string that sorts between **A** and **M** to the number 1 and any character string that sorts between **N** and **Z** to the number 2. The informat treats the unquoted range 1–3000 as a numeric range, which includes all numeric values between 1 and 3000:

```
invalue trial 'A'-'M'=1
              'N'-'Z'=2
               1-3000=3;
```

*Example 3: Create an Informat Using _ERROR_ and _SAME_*

The CHECK. informat uses _ERROR_ and _SAME_ to convert values of 1 through 4 and 99. All other values are invalid:

```
invalue check 1-4=_same_
              99=.
           other=_error_;
```

If you use a numeric informat to convert character strings that do not correspond to any values or ranges, then you receive an error message.

PICTURE Statement

Creates a template for printing numbers.

Restriction: If CASFMTLIB is specified, then the following options of the PICTURE statement are not supported: DECSEP, DIG3SEP, ROUND, DATE, TIME, DATETIME, DATETIME_UTIL and hexadecimal literals in range specifications.

Tips: As a best practice, if you specify an existing format in a value-range-set, always specify a default value by using the DEFAULT= option on page 186.

If you are using the DATATYPE= option, use the DEFAULT= option to set the default format width to be large enough to format these characters. Without setting the DEFAULT= option, the default width of a format is the width of the largest value to the right of the equation symbol.

See: *SAS Viya Formats and Informats: Reference* and *SAS Viya National Language Support: Reference Guide* for documentation about formats that are supplied by SAS.

Examples: "Example 3: Creating a Picture Format" on page 204

"Example 5: Filling a Picture Format" on page 208

"Example 10: Creating a Format in a non-English Language" on page 221

Syntax

PICTURE *name* <(*format-option(s)*)>
<*value-range-set-1* <(*picture-1-option(s)*)>
<*value-range-set-2* <(*picture-2-option(s)*)>> …>;

Summary of Optional Arguments

Control the attributes of each picture in the format

FILL='*character*'
: specifies a character that completes the formatted value.

MULTIPLIER=*n*
: specifies a number to multiply the variable's value by before it is formatted.

NOEDIT
: specifies that numbers are message characters rather than digit selectors.

PREFIX='*prefix*'
: specifies a character prefix to place in front of the formatted value.

Control the attributes of the format

DATATYPE=DATE | TIME | DATETIME | DATETIME_UTIL

enables the use of directives in the picture as a template to format date, time, or datetime values.

DECSEP='*character*'
specifies the separator character for the fractional part of a number.

DEFAULT=*length*
specifies the default length of the picture.

DIG3SEP='*character*'
specifies the three-digit separator character for a number.

FUZZ=*fuzz-factor*
specifies a fuzz factor for matching values to a range.

LANGUAGE=
specifies the language that is used for weekdays and months that you can substitute in a date, time, or datetime picture.

MAX=*length*
specifies a maximum length for the format.

MIN=*length*
specifies a minimum length for the format.

MULTILABEL
enables the assignment of labels to multiple *values-or-range* values that might have the same or overlapping values.

NOTSORTED
stores values or ranges in the order in which you define them.

ROUND
rounds the value to the nearest integer before formatting.

Control the template for printing

value-range-set
specifies one or more variable values and a template for printing those values.

Required Argument

name
names the format that you are creating.

Restriction	A user-defined format cannot be the name of a format supplied by SAS.
Requirement	The name must be a valid SAS name. A numeric format name can be up to 32 characters in length; a character format name can be up to 31 characters in length, not ending in a number. If you are creating a character format, you use a dollar sign ($) as the first character, which is why a character informat is limited to 31 characters.
Interaction	The maximum length of a format name is controlled by the VALIDFMTNAME= system option. See *SAS Viya System Options: Reference* for details.
Tip	Refer to the format later by using the name followed by a period. However, do not put a period after the format name in the VALUE statement.

Optional Arguments

DATATYPE=DATE | TIME | DATETIME | DATETIME_UTIL
enables the use of directives in the picture as a template to format date, time, or datetime values. Specify either DATE, TIME, DATETIME, or DATETIME_UTIL based on the directive that you use in the picture format. See the definition and list of directives on page 173 in the description of *picture*.

Restriction If CASFMTLIB is specified, then the DATE, TIME, DATETIME, and DATETIME_UTIL options are not supported.

Interaction DATATYPE=DATETIME results in datetime hours 00:00:00–23:59:59. DATATYPE=DATETIME_UTIL results in datetime hours between 00:00:01–24:00:00.

Tip If you format a numeric missing value, then the resulting label will be ERROR. Adding a clause to your program that checks for missing values can eliminate the ERROR label.

DEFAULT=*length*
specifies the default length of the picture. The value for DEFAULT= becomes the length of *picture* if you do not give a specific length when you associate the format with a variable.

Default The length of the longest picture value

Range 1–32767

Tip If you are using the DATATYPE= option, use the DEFAULT= option to set the default format width large enough to format these characters.

DECSEP='*character*'
specifies the separator character for the fractional part of a number.

Default . (a decimal point)

Restriction If CASFMTLIB is specified, then the DECSEP option is not supported.

DIG3SEP='*character*'
specifies the three-digit separator character for a number.

Default , (a comma)

Restriction If CASFMTLIB is specified, then the DIG3SEP option is not supported.

FILL='*character*'
specifies a character that completes the formatted value.

If the number of significant digits is less than the length of the format, then the format must complete, or fill, the formatted value:

- The format uses *character* to fill the formatted value if you specify zeros as digit selectors.
- The format uses zeros to fill the formatted value if you specify nonzero-digit selectors. The FILL= option has no effect.

If the picture includes other characters, such as a comma, which appear to the left of the digit selector that maps to the last significant digit placed, then the characters are replaced by the fill character or leading zeros.

Default ' ' (a blank)

Restriction The FILL= option is not valid when you use a function to format a value.

Interaction If you use the FILL= and PREFIX= options in the same picture, then the format places the prefix and then the fill characters.

Example "Example 5: Filling a Picture Format" on page 208

FUZZ=*fuzz-factor*

specifies a fuzz factor for matching values to a range. If a number does not match or fall in a range exactly but comes within *fuzz-factor*, on either end of the range, then the format considers it a match. For example, the following VALUE statement creates the LEVELS. format, which uses a fuzz factor of .2:

```
value levels (fuzz=.2) 1='A'
                       2='B'
                       3='C';
```

FUZZ=.2 means that if a variable value falls within .2 of a value on either end of the range, then the format uses the corresponding formatted value to print the variable value. The LEVELS. format formats the value 2.1 as **B**.

Default 1E−12 for numeric formats.

Tips Specify FUZZ=0 to save storage space when you use the VALUE statement to create numeric formats.

Use a nonzero fuzz factor only with numbers that are very close but not an exact match. If *fuzz-factor* is added to the end of one range and subtracted from the beginning of the next range, and the ranges overlap, the results can be unpredictable. A value is placed in the first range that is a match in a binary search.

A best practice is to use FUZZ=0 when you use the < exclusion operator with numeric formats.

LANGUAGE=

specifies the language that is used for weekdays and months that you can substitute in a date, time, or datetime picture. If you specify a language that is not supported or is invalid, English is used.

These are the valid values for the LANGUAGE= option:

Afrikaans	English	Macedonian	Spanish
Catalan	Finnish	Norwegian	Swedish
Croatian	French	Polish	Swiss_French
Czech	German	Portuguese	Swiss_German
Danish	Hungarian	Russian	
Dutch	Italian	Slovenian	

Defaults For single-byte character sets, the language that is specified by DFLANG= system option

For double-byte and UTF-8 character sets, the language that is specified by the LOCALE= system option

Tip To use a user-defined format in languages other than those that are supported by the LANGUAGE= option, set the LOCALE= system option to the locale for the language. In PROC FORMAT, do not specify the LANGUAGE= option. The language of a picture format is determined by the locale setting. For a list of locales, see "LOCALE= Values for PAPERSIZE and DFLANG Options" in *SAS Viya National Language Support: Reference Guide*.

See "DFLANG= System Option" in *SAS Viya National Language Support: Reference Guide*

MAX=*length*
specifies a maximum length for the format. When you associate the format with a variable, you cannot specify a width greater than the MAX= value.

Default 40

Range 1–32767

MIN=*length*
specifies a minimum length for the format.

Default 1

Range 1–32767

MULTILABEL
enables the assignment of labels to multiple *values-or-range* values that might have the same or overlapping values. The label is the formatted value that is determined by the picture definition on the right of the equal sign in a value-range-set. Here is an example of how MULTILABEL is used:

The following PICTURE statements show the two uses of the MULTILABEL option. In each case, number formats are assigned as labels. The first PICTURE statement assigns multiple labels to a single value. Multiple labels can also be assigned to a single range of values. The second PICTURE statement assigns labels to overlapping ranges of values. The MULTILABEL option enables the assignment of multiple labels to the overlapped values.

```
picture abc (multilabel)
    1000='9,999'
    1000='9999';

  picture overlap (multilabel)
    /* without decimals */
    0-999='999'
    1000-9999='9,999'

     /* with decimals */
    0-9='9.999'
    10-99='99.99'
    100-999='999.9';
```

Only multilabel-enabled procedures such as PROC MEANS and PROC SUMMARY can use multiple labels. All other procedures and the DATA step recognize only the primary label.

The *primary label* for a given entry is the formatted value (based on the picture) that is assigned to the first value or range-of-values (left side of the equal sign) that matches or contains the entry when all values (on the left side of the equal sign) are ordered sequentially. Here is an example:

- In the first PICTURE statement, the primary label for 1000 is 1,000 because the picture 9,999 is the first value that is assigned to 1000. The secondary label for 1000 is 1000, based on the 9999 picture.
- In the second PICTURE statement, the primary label for 5 is 5.000 based on the 9.999 picture that is assigned to the range 0–9 because 0–9 is sequentially the first range of values that contain 5. The secondary label for 5 is 005 because the range 0–999 occurs in sequence after the range 0–9.

Consider carefully when you assign multiple labels to a value.

Unless you use the NOTSORTED option when you assign value-range-sets, SAS stores the value-range-sets in sorted order. This order can produce unexpected results when value-range-sets with the MULTILABEL format are processed. Here is an example:

In the second PICTURE statement, the primary label for 15 is 015, and the secondary label for 15 is 15.00 because the range 0–999 occurs in sequence before the range 10–99. If you want the primary label for 15 to use the 99.99 format, then you might want to change the range 10–99 to 0–99 in the PICTURE statement. The range 0–99 occurs in sequence before the range 0–999 and will produce the desired result.

Restriction The maximum number of labels that can be created for a single format or informat is 255.

MULTIPLIER=*n*

specifies a number to multiply the variable's value by before it is formatted. The value of the MULTIPLIER= option depends both on the result of the multiplication and on the digit selectors in the *picture* portion of the *value-range-set*. For example, the following PICTURE statement creates the MILLION. format, which formats the variable value 1600000 as **`$1.6M`**:

```
picture million low-high='09.9M'
        (prefix='$' mult=.00001);
```

1600000 is first multiplied by .00001, which equals 16. Note that there is a digit selector after the decimal. The value 16 is placed into the picture beginning on the right. The value 16 overlays 09.9, and results in 01.6. Leading zeros are dropped, and the final result is 1.6M.

If the value of low-high is equal to '000M', then the result would be 16M.

Alias MULT=

Default 10^n, where *n* is the number of digits after the first decimal point in the picture. For example, suppose your data contains a value 123.456 and you want to print it using a picture of '999.999'. The format multiplies 123.456 by 10^3 to obtain a value of 123456, which results in a formatted value of 123.456.

Restriction The MULT= option is not valid when you use a function to format a value.

Examples "Example 3: Creating a Picture Format" on page 204

"Example 4: Creating a Picture Format for Large Dollar Amounts" on page 206

NOEDIT

specifies that numbers are message characters rather than digit selectors. That is, the format prints the numbers as they appear in the picture. For example, the following PICTURE statement creates the MILES. format, which formats any variable value greater than 1000 as **`>1000 miles`**:

```
picture miles 1-1000='0000'
          1000<-high='>1000 miles'(noedit);
```

Restriction The NOEDIT= option is not valid when you use a function to format a value.

NOTSORTED

stores values or ranges in the order in which you define them. If you do not specify NOTSORTED, then values or ranges are stored in sorted order by default, and SAS uses a binary searching algorithm to locate the range that a particular value falls into. If you specify NOTSORTED, then SAS searches each range in the order in which you define them until a match is found.

Use NOTSORTED if one of the following is true:

- You know the likelihood of certain ranges occurring, and you want your format to search those ranges first to save processing time.
- You want to preserve the order that you define ranges when you print a description of the format using the FMTLIB option.
- You want to preserve the order that you define ranges when you use the ORDER=DATA option and the PRELOADFMT option to analyze class variables in PROC MEANS and PROC SUMMARY.

Do not use NOTSORTED if the distribution of values is uniform or unknown, or if the number of values is relatively small. The binary searching algorithm that SAS uses when NOTSORTED is not specified optimizes the performance of the search under these conditions.

SAS automatically sets the NOTSORTED option when you use the CPORT and CIMPORT procedures to transport informats or formats between operating environments with different standard collating sequences. This automatic setting of NOTSORTED can occur when you transport informats or formats between ASCII and EBCDIC operating environments. If this situation is undesirable, then do the following:

- Use the CNTLOUT= option in the PROC FORMAT statement to create an output control data set.
- Use the CPORT procedure to create a transport file for the control data set.
- Use the CIMPORT procedure in the target operating environment to import the transport file.
- In the target operating environment, use PROC FORMAT with the CNTLIN= option to build the formats and informats from the imported control data set.

PREFIX='*prefix*'

specifies a character prefix to place in front of the formatted value. The prefix is placed in front of the value's first significant digit. You must use zero-digit selectors or the prefix is not used.

Typical uses for PREFIX= are printing leading currency symbols and minus signs. For example, the PAY. format prints the variable value 25500 as **$25,500.00**:

```
picture pay
   low-high='000,009.99' (prefix='$');
```

Default no prefix

Restriction The PREFIX= option is not valid when you use a function to format a value.

Interaction If you use the FILL= and PREFIX= options in the same picture, then the format places the prefix and then the fill characters.

Examples "Example 3: Creating a Picture Format" on page 204

"Example 5: Filling a Picture Format" on page 208

CAUTION **If the picture is not wide enough to contain both the value and the prefix, then the format truncates or omits the prefix, which results in inaccurate data.**

ROUND

rounds the value to the nearest integer before formatting. Without the ROUND option, the format multiplies the variable value by the multiplier, truncates the decimal portion (if any), and prints the result according to the template that you define. With the ROUND option, the format multiplies the variable value by the multiplier, rounds that result to the nearest integer, and then formats the value according to the template. Note that if the FUZZ= option is also specified, the rounding takes place after SAS has used the fuzz factor to determine which range the value belongs to.

Restriction If CASFMTLIB is specified, then the DECSEP option is not supported.

Tip The ROUND option rounds a value of .5 to the next highest integer.

CAUTION **The picture must be wide enough for an additional digit if rounding a number adds a digit to the number.** For example, the picture for the number .996 could be '99' (prefix '.' mult=100). After rounding the number and multiplying it by 100, the resulting number is 100. When the picture is applied, the result is .00, an inaccurate number. In order to ensure accuracy of numbers when you round numbers, make the picture wide enough to accommodate larger numbers.

value-range-set

specifies one or more variable values and a template for printing those values. *value-range-set* has the following form:

value-or-range-1 <, *value-or-range-2*, …>='*picture*'

value-or-range

See "Specifying Values or Ranges" on page 191.

picture

specifies a template for formatting values of numeric variables. The picture is a sequence of characters in single quotation marks. The maximum length for a picture is 40 characters. Pictures are specified with three types of characters: digit selectors, message characters, and directives. You can have a maximum of 16 digit selectors in a picture.

digit selectors

are numeric characters (0 through 9) that define positions for numeric values. A picture format with nonzero-digit selectors prints any leading zeros in variable values; picture digit selectors of 0 do not print leading zeros in variable values. If the picture format contains digit selectors, then a digit selector must be the first character in the picture.

Note This section uses 9s as nonzero-digit selectors.

message characters

are nonnumeric characters that are printed as specified in the picture. The following PICTURE statement contains both digit selectors (99) and message characters (**`illegal day value`**). Because the DAYS. format has nonzero-digit selectors, values are printed with leading zeros. The special range OTHER prints the message characters for any values that do not fall into the specified range (1 through 31).

```
picture days
          01-31='99'
          other='99-illegal day value';
```

Example The values 02 and 67 are printed as

```
                                   02
        67-illegal day value
```

directives

are special characters that you can use in the picture to format date, time, or datetime values.

Note: You can use directives only when you specify the DATATYPE= option on page 167 in the PICTURE statement. Ensure that the value of the DATATYPE= option is appropriate for the type of directive that you want to use. If you use an inappropriate value, the data does not format. For example, for the %a directive, use DATATYPE=DATE.

The permitted directives are as follows:

%a

abbreviated weekday name (for example, Wed).

%A

full weekday name (for example, Wednesday).

%b

abbreviated month name (for example, JAN or Jan). The character casing is determined by the SAS session locale.

Tip For the English language, to always create an abbreviated month with only an uppercase initial letter (for example, Jan), use the directive %3B.

%<*n*>B

the full month name (for example, January) if *n* is not included in the directive. *n* specifies the number of characters that appear for the month name. In comparison, the %b directive writes a three-character month abbreviation in uppercase letters for some locales.

Restrictions The *directives* argument is not supported if CASFMTLIB is specified.

n is not supported in DBCS and Unicode SAS sessions.

Example %3B would write Oct for the month of October

%C

long month name with blank padding (January through December) (for example, December).

%d

day of the month.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0d).

%e

day of the month as a two-character decimal number with leading spaces (" 1"- "31") (for example, “ 2”).

%F

full weekday name with blank padding.

%G

year as a four-digit decimal number (for example, 2008). If the week that contains January 1 has four or more days in the new year, then it is considered week 1 in the new year. Otherwise, it is the last week of the previous year and the year is considered the previous year.

%H

hour (24-hour clock).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0H).

Tip When DATETYPE=DATETIME, SAS uses datetime hours 00:00–23:59. When DATETYPE=DATETIME_UTIL, SAS uses datetime hours 00:00:01–24:00:00 and 24:00:00 is midnight at the end of the day. The hour 00:00:00 is not in the hour range and if used, converts to 24:00:00 of the previous day. When you specify DATETIME, 00:00:00 is midnight of a new day and the value 24:00:00 is midnight of the next day.

%I

hour (12-hour clock).

Alias %i

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0I).

%j
day of the year as a decimal number (1–366), with leading zero.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0j).

%m
month (1–12).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0m).

%M
minute (0–59).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0M).

%n
number of days in a duration as a decimal number (maximum of 10 digits) (for example, 25).

Restriction This directive is not valid for DBCS and Unicode SAS sessions.

%o
month (1-12) with blank padding (for example, " 2").

%p
equivalent to either a.m. or p.m.

%q
abbreviated quarter of the year string such as 1, 2, 3, or 4.

%Q
quarter of the year string, such as Quarter1, Quarter2, Quarter3, or Quarter4.

%s
fractional seconds as decimal digits (for example, .39555). The number of digits formatted is the number of digits to the right of the decimal point that is specified when you use the format. SAS rounds fractional seconds to accommodate the number of digits specified for fractional seconds.

Restriction This directive is not valid for DBCS and Unicode SAS sessions.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0s).

%S
seconds (0–59), allowing for possible leap seconds.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0S).

Example 58 and 59.07

%u
weekday as a one-digit decimal number (1–7 (Monday - Sunday)) (for example, Sunday=7).

%U
week number of the year as a decimal number (0–53). Sunday is considered the first day of the week.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0U).

%V
week number (01–53) with the first Monday as the start day of the first week. Minimum days of the first week is 4.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0SV).

%w
weekday as a one-digit decimal number (0–6 (Sunday through Saturday)) (for example, Sunday=0).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0w).

%W
week number (0–53) with the first Monday as the start day of the first week.

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0W).

%y
year without century (0–99) (for example, 93).

Note To add a leading zero before a single-digit number, insert a 0 before the directive (for example, %0y).

%Y
year with century as a four-digit decimal number (1970–2069) (for example, 1994).

%z
UTC time-zone offset.

%Z
time-zone name.

%%
the % character.

Tip Add code to your program to direct how you want missing values to be displayed.

Interaction If you specify LANGUAGE= and PICTURE= in the format definition, the format supports only English and the European languages. To use a user-defined format in languages other than those that are supported by the LANGUAGE= option, use the PICTURE= statement. Do not

specify the LANGUAGE= option. The language of a picture format is determined by the locale setting.

Details

Building a Picture Format: Step by Step

This section shows how to write a picture format that eliminates leading zeros. In the SAMPLE data set, the default printing of the variable Amount has leading zeros on numbers between 1 and –1. The PICTURE statement defines two similar formats that eliminate leading zeros on numbers between 1 and –1. The difference between the two formats is that the NOZEROSR. format specifies the ROUND option to round numbers and the NOZEROS. format does not round numbers.

This program creates, sorts, and prints the sample data set:

```
data sample;
   input Amount;
   datalines;
-2.051
-.05
-.017
  0
.093
.54
.556
6.6
14.63
0.996
-0.999
-45.00
;
run;


proc sort data=sample;
   by amount;
run;

proc print data=sample;
   title 'Default Printing of the Variable Amount';
run;
```

Default Printing of the Variable Amount

Obs	Amount
1	-45.000
2	-2.051
3	-0.999
4	-0.050
5	-0.017
6	0.000
7	0.093
8	0.540
9	0.556
10	0.996
11	6.600
12	14.630

Here is the PROC FORMAT step that creates the NOZEROSR. and NOZEROS. formats. Both formats eliminate leading zeros in the formatted values. The NOZEROSR. format specifies the ROUND option to round numbers. The NOZEROS. format does not perform rounding.

```
libname library 'SAS-library';

proc format;
   picture nozerosR (round fuzz=0)
     low -  -1      =  '000.00' (prefix='-')
     -1 < - < -.99  =  '0.99'  (prefix='-' mult=100)
     -0.99 <-<  0   =  '99'    (prefix='-.' mult=100)
                0   =  '9.99'
      0 <  -< .99   =  '99'    (prefix='.' mult=100)
      0.99 - < 1    =  '0.99'  (mult=100)
      1  - high     =  '09.99';

   picture nozeros (fuzz=0)
     low -  -1      =  '000.00' (prefix='-')
     -1 < - < -.99  =  '0.99'  (prefix='-.' mult=100)
     -0.99 <-<  0   =  '99'    (prefix='-.' mult=100)
                0   =  '9.99'
      0 <  -< .99   =  '99'    (prefix='.' mult=100)
      0.99 - < 1    =  '0.99'  (prefix='.' mult=100)
      1  - high     =  '09.99';

run;
```

The following table explains how one value from each range is formatted. For an illustration of each step, see Table 9.3 on page 181.

Table 9.2 *Building a Picture Format*

Step	Rules for Processing the PICTURE Statement	In This Example
1	Determine into which range the value falls and use that picture.	In the second range, the exclusion operator < appears on both sides of the hyphen and excludes −1 and -.99 from the range. The third range excludes 0 and .99. The fourth range excludes 1. Because exclusion operators are used, the FUZZ=0 option is specified.
2	Take the absolute value of the numeric value.	Because the absolute value is used, you need a separate range and picture for the negative numbers in order to prefix the minus sign.
3	Multiply the number by the MULT= value. If you do not specify the MULT= option, then the PICTURE statement uses the default. The default is 10^n, where *n* is the number of digit selectors to the right of the decimal * in the picture. (Step 6 discusses digit selectors further.)	Specifying a MULT= value is necessary for numbers between 0 and 1 and numbers between 0 and −1 because no decimal appears in the pictures for those ranges. Because MULT= defaults to 1, truncation of the significant digits results without a MULT= value specified. (Truncation is explained in the next step.) For the three ranges that do not have MULT= values specified, the MULT= value defaults to 100 because the corresponding picture has two digit selectors to the right of the decimal. After the MULT= value is applied, all significant digits are moved to the left of the decimal.
4	If the number is within 10^{-8} of a higher integer, round the number up. This operation is performed before the ROUND option is performed. The ROUND option is in effect. The format rounds the number after the decimal to the next highest integer if the number after the decimal is greater than or equal to .5.	Because the example uses MULT= values that ensured that all of the significant digits were moved to the left of the decimal, no significant digits are lost. The zeros are truncated. 205.1 is rounded to 205. 55.6 is rounded up to 56. 99.6 is rounded up to 100. Rounding is not performed on 5 and 660.
4a	When the ROUND option is not performed, the numbers after the decimal are truncated.	205.1 is truncated to 205. 55.6 is truncated to 55. 99.6 is truncated to 99.
5	Turn the number into a character string. If the number is shorter than the picture, then the length of the character string is equal to the number of digit selectors in the picture. Pad the character string with leading zeros. (The results are equivalent to using the Z*w*. format. Z*w*. is explained in the section on SAS formats in *SAS Viya Formats and Informats: Reference*.	205 becomes the character string 00205. 5 becomes the character string 05. 56 becomes the character string 56. 100 becomes the character string 100. 660 becomes the character string 0660. When the picture is longer than the numbers, the format adds a leading zero to the value. The format does not add leading zeros to the character string 56 and 100 because the corresponding picture has the same number of selectors.

Step	Rules for Processing the PICTURE Statement	In This Example
5a		205 becomes the character string 00205. 5 becomes the character string 05. 55 becomes the character string 55. 99 becomes the character string 099. 660 becomes the character string 0660. When the picture is longer than the numbers, the format adds a leading zero to the value. The format does not add leading zeros to the character string 55 because the corresponding picture has the same number of selectors.
6	Apply the character string to the picture. The format maps only the rightmost *n* characters in the character string, where *n* is the number of digit selectors in the picture. Thus, it is important to make sure that the picture has enough digit selectors to accommodate the characters in the string. After the format takes the right-most *n* characters, it then maps those characters to the picture from left to right. Choosing a zero or nonzero-digit selector is important if the character string contains leading zeros. If one of the leading zeros in the character string maps to a nonzero-digit selector, then it and all subsequent leading zeros and message characters become part of the formatted value. If all of the leading zeros map to zero-digit selectors, then none of the leading zeros or message characters become part of the formatted value. The format replaces the leading zeros in the character string with blanks. **	00205 is mapped to 2.05. 05 is mapped to 05. 56 is mapped to 56. 100 is mapped to 1.00. 0660 is mapped to 6.60. The leading zero is dropped from the character strings `00205` and `0660` because the leading zero maps to a zero-digit selector in the picture.

Step	Rules for Processing the PICTURE Statement	In This Example
6a		00205 is mapped to 2.05. 05 is mapped to 05. 55 is mapped to 55. 099 is mapped to 99. 0660 is mapped to 6.60. The leading zero is dropped from the character strings `00205`, `099`, and `0660`, because the leading zero maps to a zero-digit selector in the picture. The period (.) message character in the 0.99 picture is dropped because the leading zero maps to a zero-digit selector. Because the period message character is dropped, the format definition for the range 0.99 – < 1 requires a prefix of '.' in the NOZEROS. format to format a decimal number.
7	Prefix any characters that are specified in the PREFIX= option. You need the PREFIX= option because when a picture contains any digit selectors, the picture must begin with a digit selector. Thus, you cannot begin your picture with a decimal point, minus sign, or any other character that is not a digit selector.	The PREFIX= option reclaims the decimal point and the negative sign, as shown with the formatted values `-2.05`, `-.05` and `.56`.
7a		The PREFIX= option reclaims the decimal point and the negative sign, as shown with the formatted values `-2.05`, `-.05`, `.55`, and `.99`.

* A decimal in a PREFIX= option is not part of the picture.

** You can use the FILL= option to specify a character other than a blank to become part of the formatted value.

Table 9.3 *Steps to Format Various Values*

Step	Action	-2.051	-.05	.556	.996	6.6
1	Range	low – –1	–0.99 < – < 0	0 < – < .99	0.99 – < 1	1 – high
	Picture	000.00	99	99	0.99	09.99
2	Absolute value	2.051	.05	.556	.996	6.6
3	MULT=	$2.051\times10^2=205.1$	.05×100=5	.556×100=55.6	.996×100=99.6	$6.6\times10^2=660$
4	Round	205	5	56	100	660
4a	No Rounding	205	5	55	99	660

Step	Action	-2.051	-.05	.556	.996	6.6
5	Character string, rounding	00205	05	56	100	0660
5a	Character string, no rounding	00205	05	55	099	0660
6	Template, rounding	2.05	05	56	1.00	6.60
6a	Template, no rounding	2.05	05	55	99	6.60
7	Prefix, rounding	prefix='–'	prefix='–.'	prefix='.'	none	none
7a	Prefix, no rounding	prefix='–'	prefix='–.'	prefix='.'	prefix='.'	none
	Formatted result, rounding	–2.05	–.05	.56	1.00	6.60
	Formatted results, no rounding	–2.05	–.05	.55	.99	6.60

The following PROC PRINT steps associates the NOZEROSR. format and the NOZEROS. format with the AMOUNT variable in SAMPLE. The first output shows the result of rounding.

```
proc print data=sample;
   format amount nozerosr.;
   title 'Formatting the Variable Amount';
   title2 'with the NOZEROSR. Format Using Rounding';
run;

proc print data=sample;
   format amount nozeros.;
   title 'Formatting the Variable Amount';
   title2 'with the NOZEROS. Format, No Rounding';
run;
```

Formatting the Variable Amount with the NOZEROSR. Format Using Rounding

Obs	Amount
1	-45.00
2	-2.05
3	-1.00
4	-.05
5	-.02
6	00
7	.09
8	.54
9	.56
10	1.00
11	6.60
12	14.63

Formatting the Variable Amount with the NOZEROS. Format, No Rounding

Obs	Amount
1	-45.00
2	-2.05
3	-.99
4	-.05
5	-.01
6	00
7	.09
8	.54
9	.55
10	.99
11	6.60
12	14.63

CAUTION:

The picture must be wide enough for the prefix and the numbers. In this example, if the value –45.00 were formatted with NOZEROS., then the result would be 45.00 because it falls into the first range, low – –1, and the picture for that range is not wide enough to accommodate the prefixed minus sign and the number.

CAUTION:

The picture must be wide enough for an additional digit if rounding a number adds a digit to the number. For example, the picture for the number .996 could be '99' (prefix '.' mult=100). After rounding the number and multiplying it by 100, the resulting number is 100. When the picture is applied, the result is .00, an inaccurate number. In order to ensure accuracy of numbers when you round numbers, make the picture wide enough to accommodate larger numbers.

Specifying No Picture

This PICTURE statement creates a *picture-name* format that has no picture:

```
picture picture-name;
```

Using this format has the effect of applying the default SAS format to the values.

SELECT Statement

Selects entries for processing by the FMTLIB and CNTLOUT= options.

Restrictions: Only one SELECT statement can appear in a PROC FORMAT step.

You cannot use a SELECT statement and an EXCLUDE statement within the same PROC FORMAT step.

When the CASFMTLIB option is specified, the SELECT statement is not used for format libraries in CAS sessions and refers only to catalogs.

Syntax

SELECT *entry(s)*;

Required Argument

entry(s)

specifies one or more catalog entries for processing. Catalog entry names are the same as the name of the informat or format that they store. Because informats and formats can have the same name, and because character and numeric informats or formats can have the same name, you must use certain prefixes when specifying informats and formats in the SELECT statement. Follow these rules when specifying entries in the SELECT statement:

- Precede names of entries that contain character formats with a dollar sign ($).
- Precede names of entries that contain character informats with an at sign and a dollar sign (for example, @$*entry-name*).
- Precede names of entries that contain numeric informats with an at sign (@).
- Specify names of entries that contain numeric formats without a prefix.

Details

Shortcuts to Specifying Names

You can use the colon (:) and hyphen (-) wildcard characters to select entries. For example, the following SELECT statement selects all formats or informats that begin with the letter **a**.

```
select a:;
```

In addition, the following SELECT statement selects all formats or informats that occur alphabetically between **apple** and **pear**, inclusive:

```
select apple-pear;
```

How the FMTLIB and CNTLOUT= Options Affect Whether a Catalog Is Opened in Read or Update Mode

Using the FMTLIB and CNTLOUT= options in the SELECT statement indicates whether a catalog is opened for Read or Update mode. The following rules apply:

- If you use the SELECT statement and do not specify the FMTLIB option or the CNTLOUT= option, PROC FORMAT assumes that the catalog is opened in Update mode.
- If you use the SELECT statement and specify the FMTLIB option or the CNTLOUT= option, the catalog is opened for Read access.
- If you use the SELECT statement without the FMTLIB option or the CNTLOUT= option, and the SAS program does not have Write access to the catalog, the following error is written to the SAS log:

  ```
  ERROR: User does not have appropriate authorization level
         for file libref.FORMATS.CATALOG.
  ```

VALUE Statement

Creates a format that specifies character strings to use to print variable values.

See: *SAS Viya Formats and Informats: Reference* for documentation about SAS formats.

Example: "Example 6: Creating a Format for Character Values" on page 210

Syntax

VALUE <$>*name* <(*format-option(s)*)> <*value-range-set(s)*>;

Summary of Optional Arguments

DEFAULT=*length*
: specifies the default length of the format.

FUZZ=*fuzz-factor*
: specifies a fuzz factor for matching values to a range.

MAX=*length*
: specifies a maximum length for the format.

MIN=*length*
: specifies a minimum length for the format.

MULTILABEL
enables the assignment of multiple labels or external values to internal values.

NOTSORTED
stores values or ranges in the order in which you define them.

value-range-set(s)
specifies the assignment of a value or a range of values to a formatted value.

Required Argument

name
names the format that you are creating.

Restrictions The name of a user-defined format cannot be the same as the name of a format that is supplied by SAS.

Format names cannot end in a number.

Requirement The name must be a valid SAS name. A numeric format name can be up to 32 characters in length. A character format name can be up to 31 characters in length. If you are creating a character format, then use a dollar sign ($) as the first character.

Interaction The maximum length of a format name is controlled by the VALIDFMTNAME= system option. For more information, see, *SAS Viya System Options: Reference*.

Tip Refer to the format later by using the name followed by a period. However, do not use a period after the format name in the VALUE statement.

Optional Arguments

DEFAULT=*length*
specifies the default length of the format. The value for DEFAULT= becomes the length of the format if you do not give a specific length when you associate the format with a variable.

Default The length of the longest label that is assigned to the right of the equal sign

Range 1–32767

Tip As a best practice, always specify the DEFAULT= option if you specify a format as a label.

FUZZ=*fuzz-factor*
specifies a fuzz factor for matching values to a range. If a number does not match or fall in a range exactly but comes within *fuzz-factor*, then the format considers it a match. For example, the following VALUE statement creates the LEVELS. format, which uses a fuzz factor of .2:

```
value levels (fuzz=.2) 1='A'
                       2='B'
                       3='C';
```

FUZZ=.2 means that if a variable value falls within .2 of a value on either end of the range, then the format uses the corresponding formatted value to print the variable value. So the LEVELS. format formats the value 2.1 as **B**.

Default 1E−12 for numeric formats and 0 for character formats.

Tips Specify FUZZ=0 to save storage space when you use the VALUE statement to create numeric formats.

Use a nonzero fuzz factor only with numbers that are very close but not an exact match. Ranges are stored internally in sorted order (unless the NOTSORTED option is used), in order to perform a binary search. When a fuzz-factor is added to the end of one range and subtracted from the beginning of the next range, and the ranges overlap, the results can be unpredictable. A value is placed in the first range that is a match in the binary search. The exclusion operator is insufficient to override this binary search algorithm. As a best practice, when you use the exclusion operator, set FUZZ=0 or the NOTSORTED option

A best practice is to use FUZZ=0 when you use the < exclusion operator with numeric formats.

MAX=*length*
specifies a maximum length for the format. When you associate the format with a variable, you cannot specify a width greater than the MAX= value.

Default 40

Range 1–32767

MIN=*length*
specifies a minimum length for the format.

Default 1

Range 1–32767

MULTILABEL
enables the assignment of multiple labels or external values to internal values. The following VALUE statements show the two uses of the MULTILABEL option. The first VALUE statement assigns multiple labels to a single internal value. Multiple labels can also be assigned to a single range of internal values. The second VALUE statement assigns labels to overlapping ranges of internal values. The MULTILABEL option allows the assignment of multiple labels to the overlapped internal values.

```
value one (multilabel)
    1='ONE'
    1='UNO'
    1='UN';

  value agefmt (multilabel)
    15-29='below 30 years'
    30-50='between 30 and 50'
    51-high='over 50 years'
    15-19='15 to 19'
    20-25='20 to 25'
    25-39='25 to 39'
```

```
40-55='40 to 55'
56-high='56 and above';
```

Only multilabel-enabled procedures such as PROC MEANS and PROC SUMMARY can use multiple labels. All other procedures and the DATA step recognize only the primary label.

The primary label for a given entry is the external value that is assigned to the first internal value or range of internal values that matches or contains the entry when all internal values are ordered sequentially. Here is an example:

- In the first VALUE statement, the primary label for 1 is ONE because ONE is the first external value that is assigned to 1. The secondary labels for 1 are UNO and UN.
- In the second VALUE statement, the primary label for 33 is **`25 to 39`** because the range 25–39 is sequentially the first range of internal values that contains 33. The secondary label for 33 is **`between 30 and 50`** because the range 30–50 occurs in sequence after the range 25–39.

Restriction The maximum number of labels that can be created for a single format is 255.

NOTSORTED

stores values or ranges in the order in which you define them. If you do not specify NOTSORTED, then values or ranges are stored in sorted order by default, and SAS uses a binary searching algorithm to locate the range that a particular value falls into. If you specify NOTSORTED, then SAS searches each range in the order in which you define them until a match is found.

Use NOTSORTED if one of the following is true:

- You know the likelihood of certain ranges occurring, and you want your format to search those ranges first to save processing time.
- You want to preserve the order that you define ranges when you print a description of the format using the FMTLIB option.
- You want to preserve the order that you define ranges when you use the ORDER=DATA option and the PRELOADFMT option to analyze class variables in PROC MEANS or PROC SUMMARY.

Do not use NOTSORTED if the distribution of values is uniform or unknown, or if the number of values is relatively small. The binary searching algorithm that SAS uses when NOTSORTED is not specified optimizes the performance of the search under these conditions.

SAS automatically sets the NOTSORTED option when you use the CPORT and CIMPORT procedures to transport formats between operating environments with different standard collating sequences. This automatic setting of NOTSORTED can occur when you transport formats between ASCII and EBCDIC operating environments. If this situation is undesirable, then do the following:

- Use the CNTLOUT= option in the PROC FORMAT statement to create an output control data set.
- Use the CPORT procedure to create a transport file for the control data set.
- Use the CIMPORT procedure in the target operating environment to import the transport file.
- In the target operating environment, use PROC FORMAT with the CNTLIN= option to build the formats from the imported control data set.

value-range-set(s)

specifies the assignment of a value or a range of values to a formatted value. The *value-range-set(s)* have the following form:

value-or-range-1 <, *value-or-range-2*, …>=*existing-format* | [*existing-format*]

The variable values on the left side of the equal sign prints as the character string on the right side of the equal sign. The maximum length of each *value-or-range* to the left of the equal sign is 32,767 characters.

value-or-range

For details about how to specify *value-or-range*, see "Specifying Values or Ranges" on page 191.

formatted-value

specifies a character string that becomes the printed value of the variable value that appears on the left side of the equal sign. Formatted values are always character strings, regardless of whether you are creating a character or numeric format.

Formatted values can be up to 32,767 characters. For hexadecimal literals, you can use up to 32,767 typed characters, or up to 16,382 represented characters at 2 hexadecimal characters per represented character. Some procedures, however, use only the first 8 or 16 characters of a formatted value.

Requirements You must enclose a formatted value in single or double quotation marks. The following example shows a formatted value that is enclosed in double quotation marks:

```
value $ score
   'M'="Male"
   'F'="Female";
```

If a formatted value contains a single quotation mark, then enclose the value in double quotation marks:

```
value sect
   1="Smith's class"
   2="Leung's class";
```

Tip Formatting numeric variables does not preclude the use of those variables in arithmetic operations. SAS uses stored values for arithmetic operations.

existing-format

specifies a format that is supplied by SAS or an existing user-defined format. The format that you are creating uses the existing format to convert the raw data that is a match for *value-or-range* on the left side of the equal sign.

Using an existing format can be thought of as nesting formats. A nested level of one means that if you are creating the format A with the format B as a formatted value, then the procedure has to use only one existing format to create A.

Requirement If you use an existing format, then enclose the format name in square brackets (for example, [date9.]) or with parentheses and vertical bars (for example, (|date9.|)). Do not enclose the name of the existing format in single quotation marks.

Tips Avoid nesting formats more than one level. The resource requirements can increase dramatically with each additional level.

As a best practice, if you specify an existing format in *value-range-set*, always specify a default value by using the DEFAULT= option on page 186.

Examples

Example 1: Create a Format to Print Postal Codes for Selected States

The $STATE. character format prints the postal code for selected states:

```
value $state 'Delaware'='DE'
             'Florida'='FL'
             'Ohio'='OH';
```

The variable value **Delaware** prints as **DE**, the variable value **Florida** prints as **FL**, and the variable value **Ohio** prints as **OH**. Note that the $STATE. format begins with a dollar sign.

Note: Range specifications are case sensitive. In the $STATE. format above, the value **OHIO** would not match any of the specified ranges. If you are not certain what case the data values are in, then one solution is to use the UPCASE function on the data values and specify all uppercase characters for the ranges.

Example 2: Write Numeric Values as Character Values

The numeric format ANSWER. writes the values 1 and 2 as yes and no:

```
value answer 1='yes'
             2='no';
```

Example 3: Specifying No Ranges

This VALUE statement creates a *format-name* format that has no ranges:

```
value format-name;
```

Using this format has the effect of applying the default SAS format to the values.

Example 4: Create a Format Using Format as Label

This program creates the MYfmt. format to format dates based on the year:

```
data test;
  do Date='01jan2006'd to '31dec2013'd;
    do j=1 to rannor(0)*100;
       output;
    end;
  end;
run;
proc format;
   value MYfmt
        /* Format dates prior to 31DEC2011 using only a year. */
        low-'31DEC2011'd=[year4.]

        /* Format 2012 dates using the month and year. */
        '01jan2012'd-'31DEC12'd=[monyy7.]

        /* Format dates 01JAN2013 and beyond using the day, month, and year. */
```

```
        '01JAN2013'd-high=[date9.]

        /* Catch missing values. */
        other='n/a';
run;

proc freq data=test;
      table date /missing;
      format date myfmt.;
run;
```

Specifying Values or Ranges

As the syntax of the INVALUE, PICTURE, and VALUE statements indicates, you must specify values as *value-range-sets*. On the left side of the equal sign, you specify the values that you want to convert to other values. On the right side of the equal sign, you specify the values that you want the values on the left side to become. This section discusses the different forms that you can use for *value-or-range*, which represents the values on the left side of the equal sign. For details about how to specify values for the right side of the equal sign, see the "Required Arguments" section for the appropriate statement.

The INVALUE, PICTURE, and VALUE statements accept numeric values on the left side of the equal sign. In character informats, numeric ranges are treated as character strings. INVALUE and VALUE also accept character strings on the left side of the equal sign.

As the syntax shows, you can have multiple occurrences of *value-or-range* in each *value-range-set*, using a comma to separate the occurrences. Each occurrence of *value-or-range* is either one of the following:

value
a single value, such as 12 or **'CA'**. For character formats and informats, enclose the character values in single quotation marks.

You can use the keyword OTHER= as a single value. OTHER matches all values that do not match any other value or range. You cannot nest a user-defined format by using the format as the value of OTHER=, unless the format is a function that formats values. For an example, see "Example 6: Creating a Format for Character Values" on page 210.

range
a list of values (for example, 12–68 or **'A'-'Z'**). For ranges with character strings, be sure to enclose each string in single quotation marks. For example, if you want a range that includes character strings from A to Z, then specify the range as **'A'-'Z'**, with single quotation marks around the **A** and around the **Z**.

If you specify **'A-Z'**, then the procedure interprets it as a three-character string with **A** as the first character, a hyphen (-) as the second character, and a **Z** as the third character.

In numeric user-defined informats, the procedure interprets an unquoted numeric range on the left side of a *value-range-set* as a numeric range. In a character user-defined informat, the procedure interprets an unquoted numeric range on the left side of a **value-range-set** as a character string. For example, in a character informat, the range **12–86** is interpreted as **'12'–'86'**.

You can use LOW or HIGH as one value in a range, and you can use the range LOW-HIGH to encompass all values. For example, the following are valid ranges:

```
low-'ZZ'
35-high
low-high
```

In numeric ranges, LOW includes the lowest numeric value, excluding missing values. HIGH includes the largest value in the range. In character ranges, LOW includes missing values.

You can use the less than (<) symbol to exclude values from ranges. If you are excluding the first value in a range, then put the < exclusion operator after the value. If you are excluding the last value in a range, then put the < exclusion operator before the value. For example, the following range does not include 0:

```
0<-100
```

Likewise, the following range does not include 100:

```
0-<100
```

TIP When you use the < exclusion operator to place values in ranges, use the option FUZZ=0 in the VALUE statement for numeric formats. This is not necessary for character formats because FUZZ=0 is the default.

If a value at the high end of one range also appears at the low end of another range, and you do not use the < exclusion operator, then PROC FORMAT assigns the value to the first range. For example, in the following ranges, the value **AJ** is part of the first range:

```
'AA'-'AJ'=1 'AJ'-'AZ'=2
```

In this example, to include the value **AJ** in the second range, use the < exclusion operator on the first range:

```
'AA'-<'AJ'=1 'AJ'-'AZ'=2
```

If you overlap values in ranges, then PROC FORMAT returns an error message unless, for the VALUE statement, the MULTILABEL option is specified. For example, the following ranges will cause an error: **'AA'-'AK'=1 'AJ'-'AZ'=2**.

Each *value-or-range* can be up to 32,767 characters. If *value-or-range* has more than 32,767 characters, then the procedure truncates the value after it processes the first 32,767 characters.

Note: You do not have to account for every value on the left side of the equal sign. Those values are converted using the default informat or format. For example, the following VALUE statement creates the TEMP. format, which prints all occurrences of 98.6 as **NORMAL**:

```
value temp 98.6='NORMAL';
```

If the value were 96.9, then the printed result would be **96.9**.

Results: FORMAT Procedure

Output Control Data Set

The output control data set contains information that describes informats or formats. Output control data sets have a number of uses. For example, an output control data set can be edited with a DATA step to programmatically change value ranges or can be specified with a DATA step to create new formats and informats.

You create an output control data set with the CNTLOUT= option in the PROC FORMAT statement. You use output control data sets, or a set of observations from an output control data set, as an input control data set in a subsequent PROC FORMAT step using the CNTLIN= option.

Output control data sets contain an observation for every value or range in each of the informats or formats in the LIBRARY= catalog. The data set consists of variables that give either global information about each format and informat created in the PROC FORMAT step or specific information about each range and value.

The variables in the output control data set are as follows:

DATATYPE
: enables the use of directives in a picture as a template to format date, time, or datetime values.

DECSEP
: specifies the separator character for the fractional part of a number.

DEFAULT
: specifies a numeric variable that indicates the default length for format or informat.

DIG3SEP
: specifies the three-digit separator character for a number.

END
: specifies a character variable that gives the range's ending value.

EEXCL
: specifies a character variable that indicates whether the range's ending value is excluded. Valid values are as follows:

  Y
  : specifies that the range's ending value is excluded.

  N
  : specifies that the range's ending value is not excluded.

FILL
: for picture formats, specifies a numeric variable whose value is the value of the FILL= option.

FMTNAME
: specifies a character variable whose value is the format or informat name.

FUZZ
: specifies a numeric variable whose value is the value of the FUZZ= option.

HLO
specifies a character variable that contains range information about the format or informat. The following valid values can appear in any combination:

F
specifies a standard SAS format or informat that is used with a value.

H
specifies that a range's ending value is HIGH.

I
specifies a numeric informat range.

J
specifies justification for an informat.

L
specifies that a range's starting value is LOW.

M
specifies that the MULTILABEL option is in effect.

N
specifies that the format or informat has no ranges, including no OTHER= range.

O
specifies that the range is OTHER.

R
specifies that the ROUND option is in effect.

S
specifies that the NOTSORTED option is in effect.

U
specifies that the UPCASE option for an informat be used.

LABEL
specifies a character variable whose value is associated with a format or an informat.

LANGUAGE
specifies the language that is used for weekdays and months that you can substitute in a date, time, or datetime picture. If you specify a language that is not supported or is invalid, English is used.

LENGTH
specifies a numeric variable whose value is the value of the LENGTH= option.

MAX
specifies a numeric variable whose value is the value of the MAX= option.

MIN
specifies a numeric variable whose value is the value of the MIN= option.

MULT
specifies a numeric variable whose value is the value of the MULT= option.

NOEDIT
for picture formats, specifies a numeric variable whose value indicates whether the NOEDIT option is in effect. Valid values are as follows:

1
specifies that the NOEDIT option is in effect.

0
specifies that the NOEDIT option is not in effect.

PREFIX
for picture formats, specifies a character variable whose value is the value of the PREFIX= option.

SEXCL
specifies a character variable that indicates whether the range's starting value is excluded. Valid values are as follows:

Y
specifies that the range's starting value is excluded.

N
specifies that the range's starting value is not excluded.

START
specifies a character variable that gives the range's starting value.

TYPE
specifies a character variable that indicates the type of format. Possible values are as follows:

C
specifies a character format.

I
specifies a numeric informat.

J
specifies a character informat.

N
specifies a numeric format (excluding pictures).

P
specifies a picture format.

The following output shows an output control data set that contains information about all the informats and formats created in the FORMAT procedure examples.

Output 9.1 *Output Control Data Set for PROC FORMAT Examples*

Results Viewer - SAS Output

An Output Control Data Set

Obs	FMTNAME	START	END	LABEL	MIN	MAX	DEFAULT	LENGTH	FUZZ	PREFIX	MULT	FILL	NOEDIT	TYPE	SEXCL	EEXCL	HLO	DECSEP	DIG3SEP	DATATYPE	LANGUAGE
1	BENEFIT	LOW	14609	WORDDATE20.	1	40	20	20	1E-12		0.00		0	N	N	N	LF				
2	BENEFIT	14610	HIGH	** Not Eligible **	1	40	20	20	1E-12		0.00		0	N	N	N	H				
3	SALARY	LOW	HIGH	00,000,000.00	1	40	13	13	1E-12	$	100.00	*	0	P	N	N	LH	.	,		
4	USCURRENCY	LOW	HIGH	000,000	1	40	7	7	1E-12	$	1.61		0	P	N	N	LH	.	,		
5	CITY	BR1	BR1	Birmingham UK	1	40	14	14	0		0.00		0	C	N	N					
6	CITY	BR2	BR2	Plymouth UK	1	40	14	14	0		0.00		0	C	N	N					
7	CITY	BR3	BR3	York UK	1	40	14	14	0		0.00		0	C	N	N					
8	CITY	US1	US1	Denver USA	1	40	14	14	0		0.00		0	C	N	N					
9	CITY	US2	US2	Miami USA	1	40	14	14	0		0.00		0	C	N	N					
10	CITY	**OTHER**	**OTHER**	INCORRECT CODE	1	40	14	14	0		0.00		0	C	N	N	O				
11	EVALUATION	C	C	1	1	40	1	1	0		0.00		0	I	N	N					
12	EVALUATION	E	E	2	1	40	1	1	0		0.00		0	I	N	N					
13	EVALUATION	N	N	0	1	40	1	1	0		0.00		0	I	N	N					
14	EVALUATION	O	O	4	1	40	1	1	0		0.00		0	I	N	N					
15	EVALUATION	S	S	3	1	40	1	1	0		0.00		0	I	N	N					

Procedure Output

The FORMAT procedure prints output only when you specify the FMTLIB option or the PAGE option in the PROC FORMAT statement. The printed output is a table for each format or informat entry in the catalog that is specified in the LIBRARY= option. The output also contains global information and the specifics of each value or range that is defined for the format or informat. You can use the SELECT or EXCLUDE statement to control which formats and informats are represented in the FMTLIB output.

Note: The FMTLIB and PAGE options are not supported if you specify CASFMTLIB=. Specify a CAS statement with the listFormats, listFmtRanges, and listFmtValues to see information about the formats in a format library in CAS.

The FMTLIB output shown in the following output contains a description of the $CITY. format, which is created in "Example 6: Creating a Format for Character Values" on page 210.

Output 9.2 *Output from PROC FORMAT with the FMTLIB Option*

```
      FORMAT NAME: $CITY    LENGTH:   14   NUMBER OF VALUES:    6
  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:  14  FUZZ:        0
```

START	END	LABEL (VER. V7|V8 12OCT2012:11:33:26)
BR1	BR1	Birmingham UK
BR2	BR2	Plymouth UK
BR3	BR3	York UK
US1	US1	Denver USA
US2	US2	Miami USA
OTHER	**OTHER**	INCORRECT CODE

```
                  INFORMAT NAME: @EVALUATION LENGTH: 1
  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:   1  FUZZ:        0
```

START	END	INVALUE(VER. 9.4 12OCT2012:11:34:18)
C	C	1
E	E	2

```
                  INFORMAT NAME: @EVALUATION LENGTH: 1
  MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:   1  FUZZ:        0
```

START	END	INVALUE (CONT'D)
N	N	0
O	O	4
S	S	3

The fields are described below in the order in which they appear in the output, from left to right:

INFORMAT NAME or **FORMAT NAME**
the name of the informat or format. Informat names begin with an at-sign (@).

LENGTH
the length of the informat or format. PROC FORMAT determines the length in the following ways:

- For character informats, the value for LENGTH is the length of the longest raw data value on the left side of the equal sign.
- For numeric informats, the following is true:
  - LENGTH is 12 if all values on the left side of the equal sign are numeric.

- LENGTH is the same as the longest raw data value on the left side of the equal sign.

- For formats, the value for LENGTH is the length of the longest value on the right side of the equal sign.

In the output for $CITY., the LENGTH is 14 because the longest picture is 14 characters.

In the output for @EVALUATION., the length is 1 because 1 is the longest raw data value on the left side of the equal sign.

NUMBER OF VALUES
the number of values or ranges associated with the informat or format. NOZEROS. has 4 ranges, and EVAL. has 5.

MIN LENGTH
the minimum length of the informat or format. The value for MIN LENGTH is 1 unless you specify a different minimum length with the MIN= option.

MAX LENGTH
the maximum length of the informat or format. The value for MAX LENGTH is 40 unless you specify a different maximum length with the MAX= option.

DEFAULT LENGTH
the length of the longest value in the INVALUE or LABEL field, or the value of the DEFAULT= option.

FUZZ
the fuzz factor. For informats, FUZZ always is 0. For formats, the value for this field is STD if you do not use the FUZZ= option. STD signifies the default fuzz value.

START
the beginning value of a range. FMTLIB prints only the first 16 characters of a value in the START and END columns.

END
the ending value of a range. The exclusion sign (<) appears after the values in START and END, if the value is excluded from the range.

INVALUE
appears only for informats and contains the values that have informats. The SAS version specifies the version in which the informat is compatible. The date indicates the date in which the informat was created.

Note: If SAS displays version numbers V7 | V8, then the informat is compatible with those versions. If it is not compatible with earlier releases, the release that created the informat is shown. Version V9 supports long informat names (more than eight characters), and V7 | V8 do not.

LABEL
LABEL appears only for formats and contains either the formatted value or picture. The SAS version specifies the version in which the format is compatible. The date indicates the date in which the format was created.

Note: If SAS displays version numbers V7 | V8, then the format is compatible with those versions. If it is not compatible with earlier releases, the release that created the format is shown. Version V9 supports long format names (more than eight characters), and V7 | V8 do not.

For picture formats, such as NOZEROS., the LABEL section contains the PREFIX=, FILL=, and MULT= values. To note these values, FMTLIB prints the letters **P**, **F**,

and **M** to represent each option, followed by the value. For example, in the LABEL section, **P-.** indicates that the prefix value is a hyphen followed by a period.

FMTLIB prints only 40 characters in the LABEL column.

Examples: FORMAT Procedure

Example 1: Create a Format Library in a CAS Session

Features: PROC FORMAT statement option
CASFMTLIB
CAS statement

Details

This example uses the CASFMTLIB option to create a format library in a CAS session. It associates the format library with a table in the WORK directory and assigns a CAS engine libref.

Program

```
libname proclib cas;
proc format casfmtlib='myformats';
     value hospx
       1='New_York            '
       2='Massachusetts_General'
       3='Los_Angeles'
       4='Mary_Fletcher';
run;

data clinicalTrial;
  input hospital treatment $ @@;
  severity=rannor(1323)*5 + 10;
  format hospital hospx.;
  cards;
3 B  3 B  3 C   3 C
1 A  1 A  1 A   1 B
1 B  1 B  1 C   1 C
1 C  1 D  1 D   1 D
2 A  2 A  2 A   2 B
2 B  2 B  2 C   2 C
2 C  2 D  2 D   2 D
3 A  3 A  3 A   3 B
3 C  3 D  3 D   3 D
4 A  4 A  4 A   4 B
4 B  4 B  4 C   4 C
4 C  4 D  4 D   4 D
;

data proclib.clinicalTrial;
```

```
        set work.clinicalTrial;
run;

proc regselect dataproclibs.clinicalTrial;
   class treatment hospital;
   model severity=treatment hospital;
run;
```

Program Description

Create a format library in a CAS session. Assign a library with the LIBNAME statement. PROC FORMAT creates a format named *hospx*. The CASFMTLIB option specifies the name of the format library *myformats* in the CAS session.

```
libname proclib cas;
proc format casfmtlib='myformats';
     value hospx
       1='New_York            '
       2='Massachusetts_General'
       3='Los_Angeles'
       4='Mary_Fletcher';
run;
```

Associate the HOSPX format with a column or variable.

```
data clinicalTrial;
  input hospital treatment $ @@;
  severity=rannor(1323)*5 + 10;
  format hospital hospx.;
  cards;
3 B  3 B  3 C   3 C
1 A  1 A  1 A   1 B
1 B  1 B  1 C   1 C
1 C  1 D  1 D   1 D
2 A  2 A  2 A   2 B
2 B  2 B  2 C   2 C
2 C  2 D  2 D   2 D
3 A  3 A  3 A   3 B
3 C  3 D  3 D   3 D
4 A  4 A  4 A   4 B
4 B  4 B  4 C   4 C
4 C  4 D  4 D   4 D
;
```

Send actions to the CAS session. The LIBNAME statement assigns a CAS engine libref that is used to identify the table in the REGSELECT procedure step.

```
data proclib.clinicalTrial;
        set work.clinicalTrial;
run;

proc regselect dataproclibs.clinicalTrial;
   class treatment hospital;
   model severity=treatment hospital;
run;
```

Log 9.1 *Create a Format Library in a CAS Session*

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 55
 56         libname proclib cas;
 NOTE: Libref  was successfully assigned as follows:
       Engine:        CAS
       Physical Name: 5f54c1ea-4c0c-3b42-876e-ace7d5651f2b
 57         proc format casfmtlib='myformats';
 NOTE: Both CAS-based formats and catalog-based formats will be written.
 58
 58       !  value hospx 1='New_York ' 2='Massachusetts_General' 3='Los_Angeles'
 59         4='Mary_Fletcher';
 NOTE: Format HOSPX is already on the library WORK.FORMATS.
 NOTE: Format HOSPX has been output.
 60         run;

 NOTE: PROCEDURE FORMAT used (Total process time):
       real time           0.02 seconds
       cpu time            0.00 seconds
 61
 62         data clinicalTrial;
 63         input hospital treatment $ @@;
 64         severity=rannor(1323)*5 + 10;
 65         format hospital hospx.;
 66         cards;

 NOTE: SAS went to a new line when INPUT statement reached past the end of a
line.
 NOTE: The data set WORK.CLINICALTRIAL has 48 observations and 3 variables.
 NOTE: DATA statement used (Total process time):
       real time           0.01 seconds
       cpu time            0.00 seconds
 79         ;
 80
 81         data proclib.clinicalTrial;
 82         set work.clinicalTrial;
 83         run;

 NOTE: There were 48 observations read from the data set WORK.CLINICALTRIAL.
 NOTE: The data set .CLINICALTRIAL has 48 observations and 3 variables.
 NOTE: DATA statement used (Total process time):
       real time           0.07 seconds
       cpu time            0.02 seconds
 84
 85         proc regselect data=proclib.clinicalTrial;
 86         class treatment hospital;
 87         model severity=treatment hospital;
 88         run;

 NOTE: The Cloud Analytic Services server processed the request in 0.01923
seconds.
 NOTE: PROCEDURE REGSELECT used (Total process time):
       real time           0.41 seconds
       cpu time            0.10 seconds
 89
 90         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```

Creating a Format Library in a CAS Session

The output is divided into sections only for documentation appearances.

The REGSELECT Procedure

Number of Observations Read	48
Number of Observations Used	48

Class Level Information		
Class	Levels	Values
treatment	4	A B C D
hospital	4	Los_Angeles Mary_Fletcher Massachusetts_General New_York

Dimensions	
Number of Effects	3
Number of Parameters	9

Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	6	251.03651	41.83942	2.02	0.0845
Error	41	848.39550	20.69257		
Corrected Total	47	1099.43201			

Root MSE	4.54891
R-Square	0.22833
Adj R-Sq	0.11541
AIC	201.86300
AICC	205.55531
SBC	164.96141
ASE	17.67491

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	13.240102	1.737143	7.62	<.0001
treatment A	1	1.825309	1.857084	0.98	0.3314
treatment B	1	-1.711518	1.857084	-0.92	0.3621
treatment C	1	-2.357254	1.857084	-1.27	0.2115
treatment D	0	0	.	.	.
hospital Los_Angeles	1	-2.844992	1.857084	-1.53	0.1332
hospital Mary_Fletcher	1	-2.539727	1.857084	-1.37	0.1789
hospital Massachusetts_General	1	-4.498280	1.857084	-2.42	0.0199
hospital New_York	0	0	.	.	.

Task Timing		
Task	Seconds	Percent
Setup and Parsing	0.00	6.29%
Levelization	0.01	59.60%
Model Initialization	0.00	0.29%
SSCP Computation	0.00	16.01%
Model Fitting	0.00	1.69%
Cleanup	0.00	16.00%
Total	0.02	100.00%

Example 2: Create the Example Data Set

Details

Several examples in this section use the Proclib.Staff data set. In addition, many of the informats and formats that are created in these examples are stored in Library.Formats. The output data set shown in "Output Control Data Set" on page 193 contains a description of these informats and the formats.

The variables are about a small subset of employees who work for a corporation that has sites in the U.S. and Britain. The data contain the name, identification number, salary (in British pounds), location, and date of hire for each employee.

Program

```
libname proclib cas;

data proclib.staff;
   infile datalines dlm='#';
   input Name & $16. IdNumber $ Salary
         Site $ HireDate date8.;
   format hiredate date8.;
   datalines;
Capalleti, Jimmy#  2355# 21163# BR1# 30JAN13
Chen, Len#         5889# 20976# BR1# 18JUN06
Davis, Brad#       3878# 19571# BR2# 20MAR04
Leung, Brenda#     4409# 34321# BR2# 18SEP94
Martinez, Maria#   3985# 49056# US2# 10JAN93
Orfali, Philip#    0740# 50092# US2# 16FEB03
Patel, Mary#       2398# 35182# BR3# 02FEB90
```

```
Smith, Robert#     5162# 40100# BR5# 15APR06
Sorrell, Joseph#   4421# 38760# US1# 19JUN11
Zook, Carla#       7385# 22988# BR3# 18DEC10
;
```

Program Description

```
libname proclib cas;
```

Create the data set Proclib.Staff. The INPUT statement assigns the names Name, IdNumber, Salary, Site, and HireDate to the variables that appear after the DATALINES statement. The FORMAT statement assigns the standard SAS format DATE7. to the variable HireDate.

```
data proclib.staff;
   infile datalines dlm='#';
   input Name & $16. IdNumber $ Salary
         Site $ HireDate date8.;
   format hiredate date8.;
   datalines;
Capalleti, Jimmy#  2355# 21163# BR1# 30JAN13
Chen, Len#         5889# 20976# BR1# 18JUN06
Davis, Brad#       3878# 19571# BR2# 20MAR04
Leung, Brenda#     4409# 34321# BR2# 18SEP94
Martinez, Maria#   3985# 49056# US2# 10JAN93
Orfali, Philip#    0740# 50092# US2# 16FEB03
Patel, Mary#       2398# 35182# BR3# 02FEB90
Smith, Robert#     5162# 40100# BR5# 15APR06
Sorrell, Joseph#   4421# 38760# US1# 19JUN11
Zook, Carla#       7385# 22988# BR3# 18DEC10
;
```

Example 3: Creating a Picture Format

Features: PROC FORMAT statement options
- LIBRARY=

PICTURE statement options
- MULT=
- PREFIX=

LIBRARY libref

LOW and HIGH keywords

Data set: PROCLIB.STAFF from Example 1

Details

This example uses a PICTURE statement to create a format that prints the values for the variable Salary in the data set Proclib.Staff in U.S. dollars.

Program

```
libname proclib cas;

   proc format casfmtlib='proclib';
```

```
      picture uscurrency low-high='000,000' (mult=1.61 prefix='$');
   run;

proc print data=proclib.staff noobs label;

   label salary='Salary in U.S. Dollars';
   format salary uscurrency.;

   title 'proclib.STAFF with a Format for the Variable Salary';
run;
```

Program Description

Assign the SAS library reference PROCLIB.

```
libname proclib cas;
```

Specify that user-defined formats will be stored in the catalog Proclib.Formats. The LIBRARY= option specifies a SAS catalog that will contain the formats or informats that you create with PROC FORMAT. When you create the library named LIBRARY, SAS automatically creates a catalog named FORMATS inside LIBRARY.

```
   proc format casfmtlib='proclib';
```

Define the USCURRENCY. picture format. The PICTURE statement creates a template for printing numbers. LOW-HIGH ensures that all values are included in the range. The MULT= statement option specifies that each value is multiplied by 1.61. The PREFIX= statement adds a US dollar sign to any number that you format. The picture contains six digit selectors, five for the salary and one for the dollar sign prefix.

```
      picture uscurrency low-high='000,000' (mult=1.61 prefix='$');
   run;
```

Print the Proclib.Staff data set. The NOOBS option suppresses the printing of observation numbers. The LABEL option uses variable labels instead of variable names for column headings.

```
proc print data=proclib.staff noobs label;
```

Specify a label and format for the Salary variable. The LABEL statement substitutes the specific label for the variable in the report. In this case, “Salary in US Dollars” is substituted for the variable Salary for this print job only. The FORMAT statement associates the USCurrency. format with the variable name Salary for the duration of this procedure step.

```
   label salary='Salary in U.S. Dollars';
   format salary uscurrency.;
```

Specify the title.

```
   title 'proclib.STAFF with a Format for the Variable Salary';
run;
```

Output

Output 9.3 *PROCLIB.STAFF with a Format for the Variable Salary*

PROCLIB.STAFF with a Format for the Variable Salary

Name	IdNumber	Salary in U.S. Dollars	Site	HireDate
Capalleti, Jimmy	2355	$34,072	BR1	30JAN13
Chen, Len	5889	$33,771	BR1	18JUN06
Davis, Brad	3878	$31,509	BR2	20MAR04
Leung, Brenda	4409	$55,256	BR2	18SEP94
Martinez, Maria	3985	$78,980	US2	10JAN93
Orfali, Philip	0740	$80,648	US2	16FEB03
Patel, Mary	2398	$56,643	BR3	02FEB90
Smith, Robert	5162	$64,561	BR5	15APR06
Sorrell, Joseph	4421	$62,403	US1	19JUN11
Zook, Carla	7385	$37,010	BR3	18DEC10

Example 4: Creating a Picture Format for Large Dollar Amounts

Features: PICTURE statement option
MULT

Format: BIGMONEY.

Details

This example uses the MULT option of the PICTURE statement to format dollars that displays M, B, or T to indicate millions, billions, and trillions of dollars, respectively. The example uses exponential notation as well as decimal notation in the format definition.

TIP This example uses dollar values without cents and rounding is not necessary. If your dollar values include cents, you can use the ROUND option in the PICTURE statement to round values to the nearest dollar value. For more information, see "ROUND" on page 172.

Program

```
libname proclib cas;
proc format;
   picture bigmoney (fuzz=0)
      1E06-<1000000000='0000 M' (prefix='$' mult=.000001)
      1E09-<1000000000000='0000 B' (prefix='$' mult=1E-09)
      1E12-<1000000000000000='0000 T' (prefix='$' mult=1E-012);
run;

data mult;
   do i=5 to 12;
      x=16**i;
```

```
      put x=comma20. x= bigmoney.;
   end;
run;
```

Program Description

Create the BIGMONEY format. The BIGMONEY. format defines three value-range sets to format millions, billions, and trillions of dollars. 1E06 is one million, 1E09 is one billion, and 1E12 is one trillion. The < exclusion operator indicates not to include the number that follows in the range. A best practice is to use the FUZZ=0 option when you use the exclusion operator to ensure accurate numbers. For a million dollars, the range is 1,000,000 to 999,999,999. The label that is specified on the right side of the equal sign uses 4 zeros as digit selectors. The zero-digit selector specifies not to print leading zeros. The first digit selector is necessary to print the $ prefix symbol when the value is three digits. The value .000001 for the MULT= option is another way to write 1E-06, which is one millionth. Multiplying a value by the millionth, billionth, and trillionth multipliers return the number of millions, billions, and trillions of dollars.

```
libname proclib cas;
proc format;
   picture bigmoney (fuzz=0)
      1E06-<1000000000='0000 M' (prefix='$' mult=.000001)
      1E09-<1000000000000='0000 B' (prefix='$' mult=1E-09)
      1E12-<1000000000000000='0000 T' (prefix='$' mult=1E-012);
run;
```

Generate large numbers to format as dollars.

```
data mult;
   do i=5 to 12;
      x=16**i;
      put x=comma20. x= bigmoney.;
   end;
run;
```

LOG

Log 9.2 *Formatted Millions, Billions, and Trillions Dollar Amounts*

```
x=1,048,576 x=$1 M
x=16,777,216 x=$16 M
x=268,435,456 x=$268 M
x=4,294,967,296 x=$4 B
x=68,719,476,736 x=$68 B
x=1,099,511,627,776 x=$1 T
x=17,592,186,044,416 x=$17 T
x=281,474,976,710,656 x=$281 T
```

Program

```
libname proclib cas;
proc format;
   picture bigmoney (fuzz=0)
      1E06-<1000000000='0000.99 M' (prefix='$' mult=.0001)
      1E09-<1000000000000='0000.99 B' (prefix='$' mult=1E-07)
      1E12-<1000000000000000='0000.99 T' (prefix='$' mult=1E-010);
```

```
run;

data mult;
   do i=5 to 12;
      x=16**i;
      put x=comma20. x= bigmoney.;
   end;
run;
```

Program Description

In this program, the BIGMONEY. format is modified to display a more accurate number by adding decimal values.

Modify the BIGMONEY format. To display a more accurate number, the picture value and the MULT= value are modified. To display two decimal values, .99 is added to the picture. To calculate two decimal values, the value in the MULT= option is reduced from one millionth to one ten-thousandth. When 16^5 is multiplied by .0001, the results is 104.8576. The decimal values are truncated and the 104 is placed in the picture beginning on the right. The resulting formatted value is 1.04 M.

```
libname proclib cas;
proc format;
   picture bigmoney (fuzz=0)
      1E06-<1000000000='0000.99 M' (prefix='$' mult=.0001)
      1E09-<1000000000000='0000.99 B' (prefix='$' mult=1E-07)
      1E12-<1000000000000000='0000.99 T' (prefix='$' mult=1E-010);
run;
```

Generate large numbers to format as dollars.

```
data mult;
   do i=5 to 12;
      x=16**i;
      put x=comma20. x= bigmoney.;
   end;
run;
```

LOG

Log 9.3 *More Precisely Formatted Large Dollar Amounts*

```
x=1,048,576 x=$1.04 M
x=16,777,216 x=$16.77 M
x=268,435,456 x=$268.43 M
x=4,294,967,296 x=$4.29 B
x=68,719,476,736 x=$68.71 B
x=1,099,511,627,776 x=$1.09 T
x=17,592,186,044,416 x=$17.59 T
x=281,474,976,710,656 x=$281.47 T
```

Example 5: Filling a Picture Format

Features: PICTURE statement options
FILL=

PREFIX=

Details

This example does the following tasks:

- prefixes the formatted value with a specified character
- fills the leading blanks with a specified character
- shows the interaction between the FILL= and PREFIX= options

Program

```
libname proclib cas;
data pay;
   input Name $ MonthlySalary;
   datalines;
Liu   1259.45
Lars  1289.33
Kim   1439.02
Wendy 1675.21
Alex  1623.73
;

proc format;
   picture salary low-high='00,000,000.00' (fill='*' prefix='$');
run;

proc print data=pay noobs;
   format monthlysalary salary.;

   title 'Printing Salaries for a Check';
run;
```

Program Description

Create the PAY data set. The PAY data set contains the monthly salary for each employee.

```
libname proclib cas;
data pay;
   input Name $ MonthlySalary;
   datalines;
Liu   1259.45
Lars  1289.33
Kim   1439.02
Wendy 1675.21
Alex  1623.73
;
```

Define the SALARY. picture format and specify how the picture will be filled. When FILL= and PREFIX= PICTURE statement options appear in the same picture, the format places the prefix and then the fill characters. The SALARY. format fills the picture with the fill character because the picture has zeros as digit selectors. The left-most comma in the picture is replaced by the fill character.

```
proc format;
```

```
   picture salary low-high='00,000,000.00' (fill='*' prefix='$');
run;
```

Print the PAY data set. The NOOBS option suppresses the printing of observation numbers. The FORMAT statement temporarily associates the SALARY. format with the variable MonthlySalary.

```
proc print data=pay noobs;
   format monthlysalary salary.;
```

Specify the title.

```
   title 'Printing Salaries for a Check';
run;
```

Output

Output 9.4 *Printing Salaries for a Check*

Printing Salaries for a Check

Name	MonthlySalary
Liu	****$1,259.45
Lars	****$1,289.33
Kim	****$1,439.02
Wendy	****$1,675.21
Alex	****$1,623.73

Example 6: Creating a Format for Character Values

Features: VALUE statement option
OTHER

Data set: PROCLIB.STAFF

Format: USCURRENCY. from Example 2

Details

This example uses a VALUE statement to create a character format that prints a value of a character variable as a different character string.

Program

```
libname proclib cas;


   proc format casfmtlib='proclib';

      value  $city 'BR1'='Birmingham UK'
```

```
                    'BR2'='Plymouth UK'
                    'BR3'='York UK'
                    'US1'='Denver USA'
                    'US2'='Miami USA'
                    other='INCORRECT CODE';
   run;

proc print data=proclib.staff noobs label;

   label salary='Salary in U.S. Dollars';

   format salary uscurrency. site $city.;

   title 'PROCLIB.STAFF with a Format for the Variables';
   title2 'Salary and Site';
run;
```

Program Description

```
libname proclib cas;
```

Create the catalog named Proclib.Formats, where the user-defined formats will be stored.

```
   proc format casfmtlib='proclib';
```

Define the $CITY. format. The special codes BR1, BR2, and so on, are converted to the names of the corresponding cities. The keyword OTHER specifies that values in the data set that do not match any of the listed city code values are converted to the value INCORRECT CODE.

```
      value  $city 'BR1'='Birmingham UK'
                   'BR2'='Plymouth UK'
                   'BR3'='York UK'
                   'US1'='Denver USA'
                   'US2'='Miami USA'
                   other='INCORRECT CODE';
   run;
```

Print the Proclib.Staff data set. The NOOBS option suppresses the printing of observation numbers. The LABEL option uses variable labels instead of variable names for column headings.

```
proc print data=proclib.staff noobs label;
```

Specify a label for the Salary variable. The LABEL statement substitutes the label “Salary in U.S. Dollars” for the name SALARY.

```
   label salary='Salary in U.S. Dollars';
```

Specify formats for Salary and Site. The FORMAT statement temporarily associates the USCURRENCY. format with the variable SALARY and also temporarily associates the format $CITY. with the variable SITE.

```
   format salary uscurrency. site $city.;
```

Specify the titles.

```
   title 'PROCLIB.STAFF with a Format for the Variables';
   title2 'Salary and Site';
run;
```

Output

Output 9.5 *PROCLIB.STAFF with Formatted Variables for Salary and Site*

PROCLIB.STAFF with a Format for the Variables
Salary and Site

Name	IdNumber	Salary in U.S. Dollars	Site	HireDate
Capalleti, Jimmy	2355	21163	Birmingham UK	30JAN13
Chen, Len	5889	20976	Birmingham UK	18JUN06
Davis, Brad	3878	19571	Plymouth UK	20MAR04
Leung, Brenda	4409	34321	Plymouth UK	18SEP94
Martinez, Maria	3985	49056	Miami USA	10JAN93
Orfali, Philip	0740	50092	Miami USA	16FEB03
Patel, Mary	2398	35182	York UK	02FEB90
Smith, Robert	5162	40100	INCORRECT CODE	15APR06
Sorrell, Joseph	4421	38760	Denver USA	19JUN11
Zook, Carla	7385	22988	York UK	18DEC10

Example 7: Creating a Format for Missing and Nonmissing Variable Values

Features: VALUE statement
VALUE statement option
OTHER

Data set: EDUCATION

Details

The EDUCATION data set reports dropout rates and math scores for several states, and indicates a region for each state.

In this example, you use the VALUE statement to create the text value **n/a** for all math score missing values. All nonmissing math score values are formatted using the 5.1 format.

The example then prints the dropout rate and math scores for each state, by region.

Program

```
libname  cas;
options obs=20;


proc format;
   value myfmt .='n/a' other=[5.1];
run;
```

```
proc sort data=education;
   by region;
run;

proc print data=education;
   by region;
   var state dropOutRate mathScore;
   format mathScore myfmt.;
run;
```

Program Description

Set the number of observations to print.

```
libname  cas;
options obs=20;
```

Create a format for the Mathscore variable values. Use the VALUE statement to create the format MYFMT. for the Mathscore variable. When the program encounters a missing Mathscore value, the value is formatted as **n/a**. All other values for Mathscore are formatted using the 5.1 format.

```
proc format;
   value myfmt .='n/a' other=[5.1];
run;
```

Sort and print the data. Use PROC SORT to sort the data set by region. To print the data by region, specify the region variable in the PROC PRINT BY statement. To report the state, dropout rate, and math scores, use the VAR statement and specify the state, dropOutRate, and mathScore variables. Finally, use the FORMAT statement to tell SAS to format the mathScore variable using the MYFMT. format.

```
proc sort data=education;
   by region;
run;

proc print data=education;
   by region;
   var state dropOutRate mathScore;
   format mathScore myfmt.;
run;
```

Output

Output 9.6 *Dropout Rates and Math Scores for Each State in a Region*

The SAS System

Region=MW

Obs	State	DropoutRate	Math Score
1	Illinois	21.5	260.0
2	Indiana	13.8	267.0
3	Iowa	13.6	278.0
4	Kansas	17.9	n/a

Region=NE

Obs	State	DropoutRate	Math Score
5	Connecticut	16.8	270.0
6	Delaware	28.5	261.0
7	Maine	22.5	n/a
8	Maryland	26.0	260.0

Region=SE

Obs	State	DropoutRate	Math Score
9	Alabama	22.3	252.0
10	Arkansas	11.5	256.0
11	Florida	38.5	255.0
12	Georgia	27.9	258.0
13	Kentucky	32.7	256.0
14	Louisiana	43.1	246.0

Region=W

Obs	State	DropoutRate	Math Score
15	Alaska	35.8	n/a
16	Arizona	31.2	259.0
17	California	32.7	256.0
18	Colorado	24.7	267.0
19	Hawaii	18.3	251.0
20	Idaho	21.8	272.0

Example 8: Creating a Format from a Data Set

Features: PROC FORMAT statement option
CNTLIN=
Input control data set

Details

This example shows how to create a format from a SAS data set.

Here are the tasks:

- create a format from an input control data set
- create an input control data set from an existing SAS data set

Program

```
libname proclib cas;
data scale;
   input begin: $char2. end: $char2. amount: $char2.;
   datalines;
0   3    0%
4   6    3%
7   8    6%
9   10   8%
11  16   10%
;

data ctrl;
   length label $ 11;

   set scale(rename=(begin=start amount=label)) end=last;

   retain fmtname 'PercentageFormat' type 'n';

   output;

   if last then do;
      hlo='O';
      label='***ERROR***';
      output;
   end;
run;

proc print data=ctrl noobs;

   title 'The CTRL Data Set';
run;
```

Program Description

Create a temporary data set named scale. The first two variables in the data lines, called BEGIN and END, will be used to specify a range in the format. The third variable in the data lines, called AMOUNT, contains a percentage that will be used as the

formatted value in the format. Note that all three variables are character variables as required for PROC FORMAT input control data sets.

```
libname proclib cas;
data scale;
   input begin: $char2. end: $char2. amount: $char2.;
   datalines;
0   3    0%
4   6    3%
7   8    6%
9   10   8%
11  16   10%
;
```

Create the input control data set CTRL and set the length of the LABEL variable. The LENGTH statement ensures that the LABEL variable is long enough to accommodate the label ***ERROR***.

```
data ctrl;
   length label $ 11;
```

Rename variables and create an end-of-file flag. The data set CTRL is derived from WORK.SCALE. RENAME= renames BEGIN and AMOUNT as START and LABEL, respectively. The END= option creates the variable LAST, whose value is set to 1 when the last observation is processed.

```
   set scale(rename=(begin=start amount=label)) end=last;
```

Create the variables Fmtname and Type with fixed values. The RETAIN statement is more efficient than an assignment statement in this case. RETAIN retains the value of Fmtname and Type in the program data vector and eliminates the need for the value to be written on every iteration of the DATA step. Fmtname specifies the name PercentageFormat, which is the format that the input control data set creates. The Type variable specifies that the input control data set will create a numeric format.

```
   retain fmtname 'PercentageFormat' type 'n';
```

Write the observation to the output data set.

```
   output;
```

Create an "other" category. Because the only valid values for this application are 0–16, any other value (such as missing) should be indicated as an error to the user. The IF statement executes only after the DATA step has processed the last observation from the input data set. When IF executes, HLO receives a value of O to indicate that the range is OTHER, and LABEL receives a value of ***ERROR***. The OUTPUT statement writes these values as the last observation in the data set. HLO has missing values for all other observations.

```
   if last then do;
      hlo='O';
      label='***ERROR***';
      output;
   end;
run;
```

Print the control data set, CTRL. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=ctrl noobs;
```

Specify the title.

```
   title 'The CTRL Data Set';
run;
```

Output

Output 9.7 *The CTRL Data Set*

The CTRL Data Set

label	start	end	fmtname	type	hlo
0%	0	3	PercentageFormat	n	
3%	4	6	PercentageFormat	n	
6%	7	8	PercentageFormat	n	
8%	9	10	PercentageFormat	n	
10%	11	16	PercentageFormat	n	
ERROR	11	16	PercentageFormat	n	O

Store the created format in the catalog Work.Formats and specify the source for the format. The CNTLIN= option specifies that the data set CTRL is the source for the format PercentageFormat.

```
proc format library=work cntlin=ctrl;
run;
```

Create the numeric informat EVALUATION. The INVALUE statement converts the specified values. The letters O (Outstanding), S (Superior), E (Excellent), C (Commendable), and N (None) correspond to the numbers 4, 3, 2, 1, and 0, respectively.

```
proc format library=library;
   invalue evaluation 'O'=4
                      'S'=3
                      'E'=2
                      'C'=1
                      'N'=0;
run;
```

Create the WORK.POINTS data set. The instream data, which immediately follows the DATALINES statement, contains a unique identification number (EmployeeId) and bonus evaluations for each employee for each quarter of the year (Q1–Q4). Some of the bonus evaluation values that are listed in the data lines are numbers; others are character values. Where character values are listed in the data lines, the Evaluation. informat converts the value O to 4, the value S to 3, and so on. The raw data values 0 through 4 are read as themselves because they are not referenced in the definition of the informat. Converting the letter values to numbers makes it possible to calculate the total number of bonus points for each employee for the year. TotalPoints is the total number of bonus points. The addition operator is used instead of the SUM function so that any missing value will result in a missing value for TotalPoints.

```
data points;
   input EmployeeId $ (Q1-Q4) (evaluation.,+1);
   TotalPoints=q1+q2+q3+q4;
   datalines;
2355 S O O S
5889 2 . 2 2
3878 C E E E
4409 0 1 1 1
3985 3 3 3 2
0740 S E E S
2398 E E   C
5162 C C C E
4421 3 2 2 2
7385 C C C N
;
```

Example 9: Retrieving a Permanent Format

Features: PROC FORMAT statement option
LIBRARY=

Other features: FMTSEARCH= system option

Data set: SAMPLE

This example uses the LIBRARY= option and the FMTSEARCH= system option to store and retrieve a format stored in a catalog other than Work.Formats or Library.Formats. For information about the listFmtSearch and setFmtSearch actions, see *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

Program

```
libname proclib cas;

proc format casfmtlib='proclib';

picture nozeros (fuzz=0)
           low -  -1  = '000.00'(prefix='-')
       -1 < - < -.99  = '0.99' (prefix='-.' mult=100)
       -0.99 < - < 0  = '99'    (prefix='-.' mult=100)
                   0  = '0.99'
        0 < - < .99   = '99'    (prefix='.'  mult=100)
           0.99 - <1  = '0.99'  (prefix='.'  mult=100)
            1  - high = '00.99';
run;

options  fmtsearch=(proclib);

data sample;
  input Amount;
   datalines;
-2.051
-.05
-.017
  0
.093
.54
```

```
.556
6.6
14.63
0.996
-0.999
-45.00
;
run;

proc print data=sample;
   format amount nozeros.;

   title1 'Retrieving the NOZEROS. Format from PROCLIB.FORMATS';
   title2 'The SAMPLE Data Set';
run;
```

Program Description

Set up a SAS library reference named PROCLIB.

```
libname proclib cas;
```

Store the NOZEROS. format in the Proclib.Formats catalog.

```
proc format casfmtlib='proclib';
```

Create the NOZEROS. format. The PICTURE statement defines the picture format NOZEROS. See "Details" on page 177.

```
picture nozeros (fuzz=0)
           low -  -1  = '000.00'(prefix='-')
       -1 < - < -.99  = '0.99' (prefix='-.' mult=100)
       -0.99 < - < 0  = '99'    (prefix='-.' mult=100)
                   0  = '0.99'
        0 < - < .99   = '99'    (prefix='.'  mult=100)
           0.99 - <1  = '0.99'  (prefix='.'  mult=100)
            1  - high = '00.99';
run;
```

Add the Proclib.Formats catalog to the search path that SAS uses to find user-defined formats. The FMTSEARCH= system option defines the search path. The FMTSEARCH= system option requires only a libref. FMTSEARCH= assumes that the catalog name is FORMATS if no catalog name appears. Without the FMTSEARCH= option, SAS would not find the NOZEROS. format. For more information, see "FMTSEARCH= System Option" in *SAS Viya System Options: Reference*.

```
options  fmtsearch=(proclib);
```

Create the sample data set.

```
data sample;
  input Amount;
   datalines;
-2.051
-.05
-.017
  0
.093
```

```
.54
.556
6.6
14.63
0.996
-0.999
-45.00
;
run;
```

Print the SAMPLE data set. The FORMAT statement associates the NOZEROS. format with the Amount variable.

```
proc print data=sample;
   format amount nozeros.;
```

Specify the titles.

```
   title1 'Retrieving the NOZEROS. Format from PROCLIB.FORMATS';
   title2 'The SAMPLE Data Set';
run;
```

Output

Output 9.8 *Retrieving the NOZEROS. Format from PROCLIB.FORMATS*

Retrieving the NOZEROS. Format from PROCLIB.FORMATS
The SAMPLE Data Set

Obs	Amount
1	-2.05
2	-.05
3	-.01
4	0.00
5	.09
6	.54
7	.55
8	6.60
9	14.63
10	.99
11	-.99
12	-45.00

Example 10: Creating a Format in a non-English Language

Features: PICTURE statement options
DATATYPE=
LANGUAGE=

Other features: LOCALE= system option

Details

This example does the following tasks:

- Creates picture formats using directives for formatting date and datetime values by using the DATATYPE= statement option.
- Uses the LOCALE= system option to specify the locale for German.
- Prints date and datetime values to the SAS log on German using the picture formats.
- Prints a datetime value in French to the log by using the picture format that specifies LANGUAGE=French.

Program

```
libname proclib cas;
proc format;
   picture mdy(default=8) other='%0d%0m%Y' (datatype=date);
   picture langtsda (default=50) other='%A, %d %B, %Y' (datatype=date);
   picture langtsdt (default=50) other='%A, %d,%B, %Y %H %M %S'
                                 (datatype=datetime);
   picture langtsfr (default=50) other='%A, %d %B, %Y %H %M %S'
                                 (datatype=datetime language=french);
   picture alltest (default=100)
           other='%a %A %b %B %d %H %I %j %m %M %p %S %w %U %y %%'
           (datatype=datetime);
run;

option locale = de_DE;

data _null_ ;
   a= 18903;
   b = 1633239000;
   put a= mdy.;
   put a= langtsda.;
   put b= langtsdt.;
   put b= langtsfr.;
   put b= alltest.;
run ;
```

Program Description

Create formats using the PICTURE statement. Each PICTURE statement specifies the date or datetime values to format using directives. %A prints a full weekday name. %B prints a full month name. %d prints the day of the month. %Y prints the year. %H prints the hour (24–hour clock). %M prints the minute. %S prints the seconds. The first three

formats print the date or datetime in the language specified by the current value of the LOCALE= system option. The format LANGTSFT. prints the datetime in French. For the remaining directives, see the PICTURE statement on page 165.

```
libname proclib cas;
proc format;
   picture mdy(default=8) other='%0d%0m%Y' (datatype=date);
   picture langtsda (default=50) other='%A, %d %B, %Y' (datatype=date);
   picture langtsdt (default=50) other='%A, %d,%B, %Y %H %M %S'
                                 (datatype=datetime);
   picture langtsfr (default=50) other='%A, %d %B, %Y %H %M %S'
                                 (datatype=datetime language=french);
   picture alltest (default=100)
           other='%a %A %b %B %d %H %I %j %m %M %p %S %w %U %y %%'
           (datatype=datetime);
run;
```

Set the LOCALE= system option. de_DE is the locale value for Germany.

```
option locale = de_DE;
```

Print date and datetime values in German and French. The DATA step prints to the SAS log the date and datetime information for 3 October, 2011, 05:30:00 AM. All values are written in German except for the value of **b** when it is formatted using the LANGTSFR. format. The LANGSTSFR. format prints the datetime value in French.

```
data _null_ ;
   a= 18903;
   b = 1633239000;
   put a= mdy.;
   put a= langtsda.;
   put b= langtsdt.;
   put b= langtsfr.;
   put b= alltest.;
run ;
```

The SAS Log Displaying Picture Format Output in German and French

```
57 libname proclib cas;
 58
 59         proc format ;
 60
 60       !  picture mdy(default=8) other='%0d%0m%Y' (datatype=date);
 NOTE: Format MDY has been output.
 61
 61       !  picture langtsda (default=50) other='%A, %d %B, %Y' (datatype=date);
 NOTE: Format LANGTSDA has been output.
 62
 62       !  picture langtsdt (default=50) other='%A, %d,%B, %Y %H %M %S'
 63         (datatype=datetime);
 NOTE: Format LANGTSDT has been output.
 64
 64       !  picture langtsfr (default=50) other='%A, %d %B, %Y %H %M %S'
 65         (datatype=datetime language=french);
 NOTE: Format LANGTSFR has been output.
 66
 66       !  picture alltest (default=100)
 67         other='%a %A %b %B %d %H %I %j %m %M %p %S %w %U %y %%'
 68         (datatype=datetime);
 NOTE: Format ALLTEST has been output.
 69         run;

 NOTE: PROCEDURE FORMAT used (Total process time):
       real time           0.01 seconds
       cpu time            0.00 seconds



 70
 71         option locale=de_DE;
 72
 73         data _null_;
 74         a=18903;
 75         b=1633239000;
 76         put a=mdy.;
 77         put a=langtsda.;
 78         put b=langtsdt.;
 79         put b=langtsfr.;
 80         put b=alltest.;
 81         run;

 a=03102011
 a=Montag, 3 Oktober, 2011
 b=Montag, 3,Oktober, 2011 5 30 0
 b=Lundi, 3 octobre, 2011 5 30 0
 b=Mo. Montag Okt Oktober 3 5 5 276 10 30 vorm. 0 2 40 11 %
```

Example 11: Using a Format to Create a Drill-down Table

Features: VALUE statement

Other features: PROC PRINT FORMAT statement

Details

This example creates an HTML table that has population information about five U.S. states. The name of the state is a link to the state's website. The link is created using a user-defined format to format the state name. This example does the following:

- creates the data set that contains the state population information
- creates a user-defined format using the VALUE statement, where the value is an HTML link (<a>) element
- defines the name of the HTML file and the titles for the HTML file
- prints the HTML table using the user-defined format

Program

```
libname proclib cas;
data mydata;
   format population comma12.0;
   label st='State';
   label population='Population';
   input st $ 1-2 population;
   year=2000;
   datalines;
VA  7078515
NC  8049313
SC  4012012
GA  8186453
FL 15982378
;
run;

proc format;
value $COMPND
'VA'='<a href=http://www.va.gov>VA</a>'
'NC'='<a href=http://www.nc.gov>NC</a>'
'SC'='<a href=http://www.sc.gov>SC</a>'
'GA'='<a href=http://www.ga.gov>GA</a>'
'FL'='<a href=http://www.fl.gov>FL</a>';
run;

ods html file="c:\mySAS\html\Drilldown.htm"
 (title="An ODS HTML Drill-down Table Using a User-defined Format in the PRINT
 Procedure");

title h=.25in "Year 2000 U.S. Census Population";
title2 color=gray "An ODS HTML Drill-down Table Using a User-defined Format in
the PRINT Procedure";
footnote color=gray "(Click the underlined text to drill down.)";

options nodate;
proc print data=mydata label noobs;
 var st population;
 format st $compnd. ;
run;

ods html close;
ods html;
```

Program Description

Create the data set. The **mydata** DATA step creates a data set that contains information about five U.S. state populations based on the census taken in the year 2000. The variables that are created assign data for the year of the census, the state abbreviations, and the state population.

```
libname proclib cas;
data mydata;
   format population comma12.0;
   label st='State';
   label population='Population';
   input st $ 1-2 population;
   year=2000;
   datalines;
VA  7078515
NC  8049313
SC  4012012
GA  8186453
FL 15982378
;
run;
```

Create the $COMPND. format. The $COMPND. format formats each state as a link to the state's respective website.

```
proc format;
value $COMPND
'VA'='<a href=http://www.va.gov>VA</a>'
'NC'='<a href=http://www.nc.gov>NC</a>'
'SC'='<a href=http://www.sc.gov>SC</a>'
'GA'='<a href=http://www.ga.gov>GA</a>'
'FL'='<a href=http://www.fl.gov>FL</a>';
run;
```

Set up the table filename and table titles. The ODS HTML FILE= option names the directory and filename where SAS saves the HTML output.

```
ods html file="c:\mySAS\html\Drilldown.htm"
 (title="An ODS HTML Drill-down Table Using a User-defined Format in the PRINT
 Procedure");

title h=.25in "Year 2000 U.S. Census Population";
title2 color=gray "An ODS HTML Drill-down Table Using a User-defined Format in
the PRINT Procedure";
footnote color=gray "(Click the underlined text to drill down.)";
```

Print the table and close and reopen the HTML destination. The PRINT procedure uses the format $COMPND. to format the state name. The formatted name is a link to the state's respective website. The ODS HTML statements close and reopen the HTML destination so that future output does not overwrite the HTML file that you just created.

```
options nodate;
proc print data=mydata label noobs;
 var st population;
 format st $compnd. ;
run;
```

```
ods html close;
ods html;
```

Output

Output 9.9 *Using a Format to Create Drill-down Text in an HTML Table*

Year 2000 U.S. Census Population

An ODS HTML Drill-down Table Using a User-defined Format in the PRINT Procedure

State	Population
VA	7,078,515
NC	8,049,313
SC	4,012,012
GA	8,186,453
FL	15,982,378

(Cick the underlined text to drill down.)

Chapter 10

HTTP Procedure

Overview: HTTP Procedure

PROC HTTP issues Hypertext Transfer Protocol (HTTP) requests. The procedure allows an open-ended set of methods. In addition to the standard methods HEAD, TRACE, GET, POST, PUT, and DELETE, PROC HTTP accepts any method that conforms to the HTTP/1.1 standard and that is recognized by the target web server. PROC HTTP also implements HTTP/1.1 features such as persistent connections, cookie caching, EXPECT_100_CONTINUE support, and it provides authentication type specification. You can specify input data in a quoted string or you can submit it from a fileref. Custom request headers can be specified as name=value pairs in a HEADERS statement or by submitting a fully formatted input file from a fileref.

For web servers that support it, the procedure uses connection caching and cookie caching by default. You can toggle the behavior of both types of caching and clear the caches within the procedure by specifying procedure arguments. Or you turn cookie caching off by using a macro variable.

The authentication specification feature enables you to specify one or multiple authentication types for a request.

Syntax: HTTP Procedure

PROC HTTP URL="*URL-to-target*" <*option(s)*>;
  HEADERS "*HeaderName*"="*HeaderValue*" <"*HeaderName-n*"="*HeaderValue-n*">;

Statement	Task	Example
PROC HTTP	Issue HTTP requests	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10, Ex. 11
HEADERS	Specify request headers for the HTTP request	Ex. 9

PROC HTTP Statement

Invokes a web service that issues requests.

Examples:

"Example 1: A Simple GET Request" on page 236
"Example 2: A Simple PUT Request" on page 236
"Example 3: A Simple Post Request Using TLS" on page 237
"Example 4: Specifying Input Data as a String" on page 237
"Example 5: A Proxy Set in a Macro Variable" on page 238
"Example 6: A Proxy Specified in the HTTP Request" on page 239
"Example 7: A POST That Captures the Response Headers" on page 239
"Example 8: A GET That Specifies HEADEROUT_OVERWRITE" on page 240
"Example 9: A GET That Uses the HEADERS Statement" on page 241
"Example 10: A Nonstandard Method" on page 242
"Example 11: A PUT That Specifies EXPECT_100_CONTINUE" on page 243

Syntax

PROC HTTP URL="*URL-to-target*"
  <METHOD="*http-method*">
  <authentication-type-options>
  <caching-options>
  <header-options>

<*proxy-server-connection-options*>
<*web-server-authentication-options*>
<EXPECT_100_CONTINUE>
<IN="*string*" | *fileref*>
<OUT=*fileref*>
;

Summary of Optional Arguments

EXPECT_100_CONTINUE
enables a client to determine whether the target server is willing to accept the request.

IN="*string*" | *fileref*
specifies the input data.

METHOD="*http-method*"
specifies an HTTP method.

OUT=*fileref-to-response-data*
specifies a fileref where output is written.

Authenticate to Web Server

WEBPASSWORD="*basic-authentication-password*"
specifies a password for basic authentication.

WEBUSERNAME="*basic-authentication-name*"
specifies a user name for basic authentication.

Connect to Proxy Server

PROXYHOST="*proxy-host-name*"
specifies the Internet host name of an HTTP proxy server.

PROXYPASSWORD="*proxy-passwd*"
specifies an HTTP proxy server password.

PROXYUSERNAME="*proxy-user-name*"
specifies an HTTP proxy server user name.

Disable Shared Connection and Cookie Caching

CLEAR_CACHE
specifies to clear both the shared connection and cookie caches before the HTTP request is executed.

CLEAR_CONN_CACHE
specifies to clear the shared connection cache before the HTTP request is executed.

CLEAR_COOKIE_CACHE
specifies to clear the shared cookie cache before the HTTP request is executed.

NO_CONN_CACHE
disables connection caching for this procedure execution.

NO_COOKIE_CACHE
specifies cached cookies will not be used for this procedure execution.

Specify Authentication Type

AUTH_ANY

specifies that any type of authentication can be used to authenticate to the connected server.

AUTH_BASIC
specifies to use user identity authentication to authenticate to the connected server.

AUTH_NEGOTIATE
specifies to use NTLM, Kerberos. or some other type of HTTP authentication to authenticate to the connected server.

AUTH_NTLM
specifies to use NTLM authentication to authenticate to the connected server.

PROXY_AUTH_BASIC
specifies to perform user identity authentication through a proxy server.

PROXY_AUTH_NEGOTIATE
specifies to perform NTLM, Kerberos, or some other type of HTTP authentication through a proxy server.

PROXY_AUTH_NTLM
specifies to perform NTLM authentication through a proxy server.

Specify HTTP Headers

HEADEROUT_OVERWRITE
causes the response header to record only the last header block sent by the web server when a redirect occurs.

HEADEROUT=*fileref-to-response-header-file*
specifies a fileref to a text file to which the response headers are written in the format *key:value*.

Required Argument

URL="*URL-to-target*"
specifies a fully qualified URL path that identifies the endpoint for the HTTP request.

Note The URL that is passed to PROC HTTP is assumed to be URL encoded. To ensure correct encoding, use an appropriate connection class for the target web server. For example, use the AWSV4Signer class for Amazon Web Services. Or, encode reserved characters as described in RFC3986.

Tip You do not have to specify the protocol. If you set just the path (for example, `"httpbin.org"`), the actual URL used is `http://httpbin.org`.

Optional Arguments

AUTH_ANY
When a user name and password are supplied, they are used to authenticate the connected server. Otherwise, any other form of authentication that is available is used. Specifying AUTH_ANY is equivalent to specifying AUTH_NEGOTIATE, AUTH_NTLM, and AUTH_BASIC on the procedure statement.

Default This is the default authentication type if an authentication type is not specified.

Tip Since there is a chance of more than one trip to the HTTP server, specify EXPECT_100_CONTINUE to prevent data from being uploaded multiple times.

AUTH_BASIC
specifies to use user identity authentication to authenticate the connected server. The user name and password are supplied with the WEBUSERNAME and WEBPASSWORD arguments.

AUTH_NTLM
specifies to use NTLM authentication to authenticate to the connected server. As long as your current user identity has permissions, authentication is established.

Restriction NTLM is currently available only on Windows clients.

AUTH_NEGOTIATE
specifies to use NTLM, Kerberos. or some other type of HTTP authentication to authenticate to the connected server. As long as your current user identity has permissions, authentication is established.

CLEAR_CACHE
specifies to clear both the shared connection and cookie caches before the HTTP request is executed.

CLEAR_CONN_CACHE
specifies to clear the shared connection cache before the HTTP request is executed.

CLEAR_COOKIE_CACHE
specifies to clear the shared cookie cache before the HTTP request is executed.

EXPECT_100_CONTINUE
enables a client that is sending a request message with a request body to determine whether the target server is willing to accept the request, based on the request headers. Use EXPECT_100_CONTINUE when you are sending large amounts of data and want to make sure that no unnecessary transfers of the data occur. For more information, see http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.2.3.

Valid in HTTP requests that specify the IN= argument, most commonly with PUT.

Interaction This argument is used in conjunction with the HEADEROUT= argument.

Example "Example 11: A PUT That Specifies EXPECT_100_CONTINUE" on page 243

HEADEROUT=*fileref-to-response-header-file*
specifies a fileref to a text file to which the response headers are written in the format *key:value*.

Examples "Example 7: A POST That Captures the Response Headers" on page 239

"Example 11: A PUT That Specifies EXPECT_100_CONTINUE" on page 243

HEADEROUT_OVERWRITE
used in conjunction with the HEADEROUT= argument, causes the response header to record only the last header block sent by the web server when a redirect occurs.

Example "Example 8: A GET That Specifies HEADEROUT_OVERWRITE" on page 240

IN="*string*" | *fileref*
specifies the input data. You can specify input data in a quoted string or in a fileref.

Requirement This option is required when the POST and PUT methods are used.

Example "Example 4: Specifying Input Data as a String" on page 237

METHOD="*http-method*"
specifies an HTTP method. Any method that conforms to the HTTP/1.1 standard and is recognizable by the target web server is acceptable. For information, see the HTTP/1.1 specification at www.w3.org.

Default If you omit the METHOD argument and do not specify the IN argument, the default method is GET. If you omit METHOD and do specify the IN argument, the default method is POST.

Examples "Example 2: A Simple PUT Request" on page 236

"Example 9: A GET That Uses the HEADERS Statement" on page 241

"Example 10: A Nonstandard Method" on page 242

NO_CONN_CACHE
disables connection caching for this HTTP request. The connection will be made with the specified connection parameters.

NO_COOKIE_CACHE
specifies cached cookies will not be used for this HTTP request. This argument does not prevent cookies from being sent manually with the "Cookie" header.

OUT=*fileref-to-response-data*
specifies a fileref that indicates where output is written.

Example "Example 2: A Simple PUT Request" on page 236

PROXY_AUTH_BASIC
specifies to perform user identity authentication through a proxy server. The user name and password are supplied with the PROXYUSERNAME and PROXYPASSWORD arguments.

PROXY_AUTH_NTLM
specifies to perform NTLM authentication through a proxy server. As long as your current user identity has permissions, authentication is established.

Restriction NTLM is currently available only on Windows clients.

PROXY_AUTH_NEGOTIATE
specifies to perform NTLM, Kerberos, or some other type of HTTP authentication through a proxy server. As long as your current user identity has permissions, authentication is established.

PROXYHOST="*proxy-host-name*"
specifies the Internet host name of an HTTP proxy server. A name in the following form is recommended:

```
protocol://host-name:port-number
```

If the protocol is omitted from the name, the default protocol is **`http://`**. If a port number is not specified, the default port number is **`80`**.

Example "Example 6: A Proxy Specified in the HTTP Request" on page 239

PROXYPASSWORD="*proxy-passwd*"
specifies an HTTP proxy server password.

Tips The password is required only if your proxy server requires credentials.

Encodings that are produced by PROC PWENCODE are supported.

PROXYUSERNAME="*proxy-user-name*"
specifies an HTTP proxy server user name.

Tip The user name is required only if your proxy server requires credentials.

WEBPASSWORD="*basic-authentication-password*"
specifies a password for basic authentication.

Tip Encodings that are produced by PROC PWENCODE are supported.

WEBUSERNAME="*basic-authentication-name*"
specifies a user name for basic authentication.

HEADERS Statement

Specifies request headers for the HTTP request.

Supports: All HTTP methods

Example: "Example 9: A GET That Uses the HEADERS Statement" on page 241

Syntax

HEADERS "*HeaderName*"="*HeaderValue*" <"*HeaderName-n*"="*HeaderValue-n*">;

Required Argument

"HeaderName"="HeaderValue"
is a name and value pair that represents a header name and its value. The HeaderName can be a standard header name or a custom header name. For information about header field definitions, see the HTTP/1.1 specification at www.w3.org.

Note: Do not specify a colon (:) in the header name. The name=value pairs are automatically translated into the following form:

```
HeaderName : HeaderValue
```

Details

The HEADERS statement enables you to specify header values easily within the procedure request, instead of having to provide a fully formatted input file via a fileref. Use the HEADERS statement to specify the content-type and character set of the document that you are uploading when the values are different from the default values for the method.

HTTP Method	Default Content-Type
POST	application/x-www-form-urlencoded
PUT	application/octet-stream

Using Hypertext Transfer Protocol Secure (HTTPS)

HTTP Security: TLS and Data Encryption

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), enables web browsers and web servers to communicate over a secured connection by encrypting data. Both browsers and servers encrypt data before the data is transmitted. The receiving browser or server then decrypts the data before it is processed.

Note: All discussion of TLS is also applicable to the predecessor protocol, SSL.

Making PROC HTTP Calls Using the HTTPS Protocol

Secure communication over HTTP (HTTPS) is controlled by either the System Trusted Root CA or by a Trusted Root CA bundle that is configured using the SSLCALISTLOC system option. For more information, see "SSLCALISTLOC= System Option" in *Encryption in SAS Viya*. "Example 3: A Simple Post Request Using TLS" on page 237 shows how the system option is specified in a PROC HTTP request.

Using Authentication Other Than Basic

The ability to specify the authentication type is useful when you know which authentication type is required for a request to succeed in advance. Specifying the correct type, rather than requiring the procedure to negotiate it, optimizes procedure execution. For example, if you know that the server supports only Kerberos authentication, it is a good idea to specify the AUTH_NEGOTIATE argument. If you know that the server supports only NTLM authentication, then specify AUTH_NTLM.

If you do not specify the authentication type, the default type is AUTH_ANY. AUTH_ANY is equivalent to specifying AUTH_NTLM, AUTH_NEGOTIATE, and AUTH_BASIC together in the request. AUTH_NTLM authentication is attempted first (for Windows only), then AUTH_NEGOTIATE, and so on. The server ultimately determines which authentication type is used. If the server that you are connecting to supports the NTLM authentication protocol or the Kerberos authentication protocol, it usually is not necessary to specify a user name and password. As long as your current user identity has permissions, authentication is established.

EXPECT_100_CONTINUE support is provided to optimize requests that must make more than one trip to the server. This argument prevents the data from being uploaded multiple times.

When using AUTH_NEGOTIATE and AUTH_NTLM, do not turn off connection caching. These authentication types require connection caching to be enabled.

Wire Logging

Wire logging logs packets of information as they appear on the network. This information is normally referred to as a dump. Wire dumps enable you to see what information is being sent to the server and what information the server is sending back. Because you can see the raw data, wire dumps can be useful in debugging your programs.

Logger APP.TK.HTTPC is used to log HTTP-specific messages. The wire dumps that the logger generates can be enabled by setting logger APP.TK.HTTPC to the DEBUG level or higher. At the DEBUG level, the first 64 bytes of incoming and outgoing data is logged. At the TRACE level, all of the data is written to the log. Note that at the TRACE level, performance can be greatly diminished.

Using Encodings with PROC HTTP

Responses are not encoded to session encodings. You must supply the request with the encoding that you want to use, and set the content type.

PROC HTTP Macro Variables

PROC HTTP produces the following automatic macro variables to enable you to set or change default PROC HTTP settings.

PROCHTTP_PROXY="*proxy-server-name-and-port-number*";
: sets a default proxy server for PROC HTTP requests. Once set, the specified proxy server establishes a proxy for all PROC HTTP requests in the SAS session, unless you specify the PROXYHOST= argument in the PROC HTTP request. The value that is specified in the procedure argument overrides the value that is specified in the macro variable. Specify the PROXYHOST= argument with a value that is different from the macro variable to use a different proxy server for a request. Specify PROXYHOST= without a value to disable proxy use for a request. For more information, see "Example 5: A Proxy Set in a Macro Variable" on page 238.

PROCHTTP_NOCOOKIES= blank | *integer*;
: provides global control of cookie caching for PROC HTTP requests. Omitting the macro variable or specifying the macro variable without a value enables cookie caching (cookie caching is on by default). To globally disable cookie caching, specify a nonzero value in the macro variable.

The macro variables are set with the %LET statement. The following is an example of a statement that globally disables cookie caching:

```
%let PROCHTTP_NOCOOKIES=1;
```

When you disable cookie caching, you can delete the macro variable from the symbol table with the %SYMDEL statement.

Examples: HTTP Procedure

Example 1: A Simple GET Request

Features: METHOD= Argument
URL= Argument
OUT= Argument

Details

This example makes a GET request. GET is the simplest and most common request that you can make with PROC HTTP.

Program

```
filename resp TEMP;

proc http
   method="GET"
   url="http://httpbin.org/get"
   out=resp;
run;
```

Example 2: A Simple PUT Request

Features: IN= Argument
OUT= Argument
HEADEROUT Statement

Details

This example makes a simple PUT method call to a server on the local network. The file to upload is identified by a fileref in the IN= argument. The response and the output headers are written to filerefs.

Program

```
filename resp TEMP;
filename headout TEMP;

filename input "fileToUpload.data";

proc http
   method="PUT"
   url="http://httpbin.org/put"
```

```
   in=input
   out=resp
   headerout=headout;
run;
```

Example 3: A Simple Post Request Using TLS

Features: SSLCALISTLOC= System Option

Details

This example makes a POST request to a server that uses TLS. The SSLCALISTLOC= system option configures the certificate that is used to make the trusted connection.

Note: You can also add the SSLCALISTLOC= system option to the sasv9.cfg file that is loaded when the SAS session starts.

Program

```
options set= SSLCALISTLOC="path-to-trustedcerts.pem";

filename out "u:\prochttp\Testware\Test_out.txt";

proc http
   url="http://httpbin.org/post"
   method="POST"
   in="text to write out"
   out=out;
run;
```

Example 4: Specifying Input Data as a String

Features: IN= "*string*"

Details

The PROC HTTP IN= argument accepts a quoted input string or a fileref to submit input data. Specifying input in a string makes it easier to send text posts and form-based posts. This example submits the form that can be found at http://httpbin.org/forms/post. The response is written to a response file.

Program

```
filename resp TEMP;

proc http
   url="http://httpbin.org/post"
   in='custname=Sas+User&custtel=919-555-5555&custemail=sas.user%40
   sas.com&size=medium&topping=cheese&delivery=12%3A00&comments=Dont+Drop+It'
   out=resp;
run;
```

```
data _null_;
   infile resp;
   input;
   put _infile_;
run;
```

This is the content of the Resp file:


```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "comments": "Dont Drop It",
    "custemail": "sas.user@sas.com",
    "custname": "Sas User",
    "custtel": "919-555-5555",
    "delivery": "12:00",
    "size": "medium",
    "topping": "cheese"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "133",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "SAS/9",
  },
  "json": null,
  "origin": "149.173.1.80, 104.129.194.85",
  "url": "http://httpbin.org/post"
}
```


Example 5: A Proxy Set in a Macro Variable

Features: PROCHTTP_PROXYHOST= Macro Variable
IN= "*string*"

Details

This example makes a method call to an external server and, therefore, requires the use of a proxy server. The proxy server's Internet host name and port number are specified in the PROCHTTP_PROXYHOST macro variable. Because the proxy is set in the macro variable, it is available to all subsequent HTTP requests that are made in the SAS session. Parameters to the POST are read from a text string that is specified in the IN= argument. The response is written to fileref OUT.

Program

```
%let PROCHTTP_PROXY="http://myproxy:889";

filename out "u:\prochttp\Testware\ProxyTest_out.txt";

proc http
   url="http://httpbin.org/post"
```

```
   method="post"
   in="text to write out"
   out=out;
run;
```

Example 6: A Proxy Specified in the HTTP Request

Features: IN="*string*"
PROXYHOST= Argument

Details

This example uses the PROXYHOST argument to specify the proxy to connect to an external server. The value that is specified in the PROXYHOST argument takes precedence over the value in the PROCHTTP_PROXYHOST= macro variable, if it is set. The global proxy is used for subsequent HTTP requests, unless the PROXYHOST argument is specified again.

The example makes the same request as in "Example 5: A Proxy Set in a Macro Variable" on page 238.

Program

```
%let PROCHTTP_PROXY="http://myproxy:889";

filename out "u:\prochttp\Testware\ProxyTest_out.txt";

proc http
   url="http://httpbin.org/post"
   method="post"
   in="text to write out"
   out=out
   proxyhost="http://myproxy2:776";
run;
```

Example 7: A POST That Captures the Response Headers

Features: IN="*string*"
HEADEROUT= Argument

Details

This example makes the same POST request as in "Example 5: A Proxy Set in a Macro Variable" on page 238 but captures the response headers in a file called headerOut.txt.

Program

```
%let PROCHTTP_PROXY="http://myproxy:889";

filename out "u:\prochttp\Testware\ProxyTest_out.txt";
filename hdrout "u:\prochttp\Testware\headerOut.txt";
```

```
 proc http
   url="http://httpbin.org/post"
   method="post"
   in="text to write out"
   out=out
   headerout=hdrout;
run;
```

Example 8: A GET That Specifies HEADEROUT_OVERWRITE

Features: HEADEROUT argument
HEADEROUT_OVERWRITE argument

Details

This example shows the effects of the HEADEROUT_OVERWRITE argument. The GET requests redirect twice before reaching their destination. HEADEROUT_OVERWRITE causes only the last output header to be recorded.

Example of Normal HEADEROUT Output After a Redirect

```
filename hdrs "u:\prochttp\Testware\GetHdr_out.txt";
filename out "u:\prochttp\Testware\GetTest_out.txt";

proc http
   url="http://httpbin.org/redirect/2"
   method="GET"
   headerout=hdrs
   out=out;
run;
```

This is the content of GetHdr_out.txt:

```
HTTP/1.1 302 FOUND
Server: nginx
Date: Mon, 20 Apr 2015 14:19:52 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 247
Connection: keep-alive
Location: /relative-redirect/1
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

HTTP/1.1 302 FOUND
Server: nginx
Date: Mon, 20 Apr 2015 14:19:53 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 0
Connection: keep-alive
Location: /get
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 20 Apr 2015 14:19:53 GMT
Content-Type: application/json
Content-Length: 195
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

Example of HEADEROUT Request with HEADEROUT_OVERWRITE

```
filename hdrs "u:\prochttp\Testware\GetHdr2_out.txt";
filename out "u:\prochttp\Testware\GetTest2_out.txt";

proc http
 url="http://httpbin.org/redirect/2"
method="GET"
headerout=hdrs
out=out
HEADEROUT_OVERWRITE;
run;
```

This is the content of GetHdr2_out.txt:

```
HTTP/1.1 200 OK
Server: nginx
Date: Mon, 20 Apr 2015 14:22:48 GMT
Content-Type: application/json
Content-Length: 195
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

Example 9: A GET That Uses the HEADERS Statement

Features: HEADERS Statement
GET Method

Details

The following is an example of a GET method request that specifies the HEADERS statement. GET is the default method when the IN argument is not specified.

Program

```
filename resp TEMP;

proc http
   url="http://httpbin.org/headers"
   out=resp;
 headers
   "Accept"="application/json";
run;

data _null_;
   infile resp;
   input;
   put _infile_;
run;
```

The output looks like this:

```
  "headers": {
    "Accept": "*/*,application/json",
    "Host": "httpbin.org",
    "User-Agent": "SAS/9",
  }
}
```

Example 10: A Nonstandard Method

Features: METHOD Argument

Details

This example submits the MKCOL WEBDAV http method. Output is written to a temporary file named Resp. There are no input and output requirements for nonstandard methods. As long as the target server returns data and you have specified a valid OUT, data will be written to your OUT fileref. Here, output is written to Resp.

Program

```
filename resp TEMP;

proc http
   url="http://hostname/directory/"
   method="MKCOL"
   out=resp;
run;
```

Example 11: A PUT That Specifies EXPECT_100_CONTINUE

Features: EXPECT_100_CONTINUE Argument
HEADEROUT= Argument

Details

This example specifies the EXPECT_100_CONTINUE header.

Program

```
filename resp TEMP;
filename hdrs TEMP;

proc http
   url="http://httpbin.org/put"
   method="PUT"
   in='Some Put Data'
   out=resp
   headerout=hdrs
   EXPECT_100_CONTINUE;
run;

data _null_;
   infile hdrs;
   input;
   put _infile_;
run;

data _null_;
   infile resp;
   input;
   put _infile_;
run;
```

The output in the HDRS looks like this:

```
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Server: gunicorn/18.0
Date: Mon, 24 Nov 2014 20:18:29 GMT
Content-Type: application/json
Content-Length: 652
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-Cache: MISS from transproxy
Via: 1.1 vegur, 1.1 transproxy (squid)
Connection: keep-alive
```

The output in the Resp file looks like this:

```
{
  "args": {},
  "data": "Some Put Data",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "13",
    "Content-Type": "application/octet-stream",
    "Host": "httpbin.org",
    "User-Agent": "SAS/9",
    "Xxpect": "100-continue",
  },
  "json": null,
  "origin": "149.173.1.80, 104.129.194.85",
  "url": "http://httpbin.org/put"
}
```

Chapter 11

IMPORT Procedure

Overview: IMPORT Procedure

What Does the IMPORT Procedure Do?

The IMPORT procedure reads data from an external data source and writes it to a SAS data set. In SAS, you can import JMP files and delimited files.

SAS Viya supports only the UTF-8 encoding. For information about the encoding of your data sets in SAS Viya, see *Migrating Data to UTF-8 for SAS Viya* and *SAS Viya FAQ for Processing UTF-8 Data*.

In delimited files, a delimiter (such as a blank, comma, or tab) separates columns of data values. If you license SAS/ACCESS Interface to PC Files, additional external data sources can include Microsoft Excel files, and Lotus spreadsheets. For more information, see *SAS/ACCESS Interface to PC Files for SAS Viya: Reference*.

In SAS, you can import data from JMP 7 or later files, and JMP variables can be up to 255 characters long. You can also import value labels to a SAS format catalog. Extended

attributes are now used automatically, and the META= statement is no longer supported. For more information, see "JMP Files" in *SAS/ACCESS Interface to PC Files for SAS Viya: Reference*.

Format Catalog Encodings in SAS Viya

SAS Viya supports only the UTF-8 encoding.

For more information about the encodings of format catalogs, see *Migrating Data to UTF-8 for SAS Viya* and *SAS Viya FAQ for Processing UTF-8 Data*.

Support for the VARCHAR Data Type

PROC IMPORT supports the VARCHAR data type in CAS. VARCHAR stores a character variable that can have a varying length. The length that you specify for the variable represents the maximum number of characters that you want to store.

The VARCHAR data type is similar to the CHAR data type. CHAR variables have a length that is measured in terms of bytes. VARCHAR variables have a length that is measured in terms of characters rather than bytes. For information about using VARCHAR, see "Data Types Supported in the CAS DATA Step" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

In the following example, the CAS engine is used with the LENGTH statement to create a VARCHAR variable and a CHAR variable. The VARCHAR variable, X, has a length of 30 and the CHAR variable, Y, also has a length of 30.

```
libname mycas cas;
data mycas.string;
   length x varchar(30);
   length y $30;
   x = 'abc'; y = 'def';
run;
proc contents data=mycas.string; run;
```

Here is the output that the code produces.

The CONTENTS Procedure

Data Set Name	MYCAS.STRING	Observations	1
Member Type	DATA	Variables	2
Engine	CAS	Indexes	0
Created	09/11/2016 13:48:30	Observation Length	48
Last Modified	09/11/2016 13:48:30	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Limit	100MB

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	x	Varchar	30
2	y	Char	30

Syntax: IMPORT Procedure

Restrictions: The IMPORT procedure is available for the Linux operating environments.

A pathname for a file can have a maximum length of 201 characters.

Interaction: All data with the percent sign (%) is considered character data to avoid misinterpretation. Percentage data is considered character data because of the danger of misinterpretation.

Supports: PROC IMPORT supports the CSV, TAB, DLM, and JMP file types in CAS.

Note: You can use PROC IMPORT to import an external file to a SAS data set or to a CAS table.

See: "ANYDTDTM Informat" in *SAS Viya Formats and Informats: Reference*

SAS Viya supports only the UTF-8 encoding. For information about the encoding of your data sets in SAS Viya, see *Migrating Data to UTF-8 for SAS Viya* and *SAS Viya FAQ for Processing UTF-8 Data*.

For information about using VARCHAR, see "Data Types Supported in the CAS DATA Step" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

PROC IMPORT

DATAFILE="*filename*" | TABLE="*tablename*"

OUT=<*libref*.>*SAS data set* <(*SAS data set option(s)*)>

<DBMS=*identifier*> <REPLACE>;

statements for importing from delimited files

DATAROW=*n*;

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

GETNAMES=YES | NO;

GUESSINGROWS=*n* | MAX;

statements for importing from JMP files

DBENCODING=*12-char SAS encoding-value*;

FMTLIB=<*libref.*>*format-catalog*;

Statement	Task	Example
PROC IMPORT	Import an external data file to a SAS data set	Ex. 1, Ex. 2, Ex. 3, Ex. 4
DATAROW	Start reading data from a specific row in the delimited text file	Ex. 3
DBENCODING	Indicate the encoding character set to use for the JMP file	
DELIMITER	Specify the delimiter that separates columns of data in the input file	Ex. 1, Ex. 3, Ex. 4
FMTLIB	Save value labels to the specified SAS format catalog	
GETNAMES	Generate SAS variable names from the data values in the first row in the input file	Ex. 1, Ex. 2
GUESSINGROWS	Specify the number of rows of the input file to scan to determine the appropriate data type and length for the variables	

PROC IMPORT Statement

Imports an external data file to a SAS data set.

Syntax

PROC IMPORT

DATAFILE="*filename* " | TABLE="*tablename* "

OUT=<*libref .*>*SAS data set* <(*SAS data set option(s)*)>

<DBMS=*identifier*> <REPLACE>;

Summary of Optional Arguments

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

REPLACE
: overwrites an existing SAS data set.

SAS data set option(s)
: specifies SAS data set options.

Required Arguments

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. For more information about the FILENAME statement, see *SAS Viya Statements: Reference*. For more information about PC file formats, see *SAS/ACCESS Interface to PC Files for SAS Viya: Reference*

If you specify a fileref or if the complete path and filename does not include special characters such as the backslash in a path, lowercase characters, or spaces, 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 a SAS DATA step with an INFILE statement that includes the RECFM=F and LRECL= options. For more information, see the INFILE statement.

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 that the IMPORT procedure supports is 32767.

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.

Examples

"Example 1: Importing a Delimited File" on page 255

"Example 2: Importing a Specific Delimited File Using a Fileref" on page 259

"Example 3: Importing a Tab-Delimited File" on page 263

"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 266

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. For more information, see "Rules for SAS Data Set Names, View Names, and Item Store Names" in *SAS Language Reference: Concepts*.

Examples "Example 1: Importing a Delimited File" on page 255

"Example 2: Importing a Specific Delimited File Using a Fileref" on page 259

"Example 3: Importing a Tab-Delimited File" on page 263

"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 266

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.

Requirements You must have a license for SAS/ACCESS Interface to PC Files to import to a DBMS table.

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

Note Use TABLE= for Microsoft Access database files.

Optional Arguments

DBMS=*identifier*
specifies the type of data to import. You can import delimited files or JMP files (DBMS=JMP) in SAS. The JMP file format must be Version 7 or later, and JMP variable names can be up to 255 characters long. SAS supports importing JMP files that have more than 32,767 variables.

To import a tab-delimited file, specify TAB as the identifier. To import any other delimited file that does not end in .CSV, specify DLM as the identifier. For a comma-separated file with a .CSV extension, DBMS= is optional. The IMPORT procedure recognizes .CSV as an extension for a comma-separated file.

See Table 7.1 on page 102 for more information about identifiers for this option.

Examples "Example 1: Importing a Delimited File" on page 255

"Example 2: Importing a Specific Delimited File Using a Fileref" on page 259

"Example 3: Importing a Tab-Delimited File" on page 263

"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 266

REPLACE

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

CAUTION:

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

If you write to an existing generation data set using the IMPORT procedure with the REPLACE option and you do one of the following:

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

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

Examples "Example 1: Importing a Delimited File" on page 255

"Example 2: Importing a Specific Delimited File Using a Fileref" on page 259

"Example 3: Importing a Tab-Delimited File" on page 263

"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 266

SAS data set option(s)

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 option. 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.

See *SAS Viya Data Set Options: Reference*

DATAROW Statement

Starts reading data from the specified row number in the delimited text file.

Default: When GETNAMES=NO: 1, when GETNAMES=YES: 2

Restriction: When GETNAMES=NO, DATAROW must be equal to or greater than 1. When GETNAMES=YES, DATAROW must be equal to or greater than 2.

Interaction: The DATAROW statement is valid only for delimited files.

See: "GETNAMES Statement" on page 253

Example: "Example 3: Importing a Tab-Delimited File" on page 263

Syntax

DATAROW=*n*;

Required Argument

n
specifies the row number in the input file for the IMPORT procedure to start reading data.

DBENCODING Statement

Indicates the encoding character set to use for the JMP file.

Interaction: The DBENCODING statement is valid only when DBMS=JMP.

Syntax

DBENCODING=*12-char SAS encoding-value*;

Required Argument

12-char SAS encoding-value
indicates the encoding to use with JMP 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. This value can have a maximum length of 12 characters.

DELIMITER Statement

Specifies the delimiter that separates columns of data in the input file.

Default: Blank space

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

Example: "Example 1: Importing a Delimited File" on page 255

Syntax

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

Required Argument

***char* | '*nn*'x**
specifies the delimiter that separates columns of data in the input file. You can specify the delimiter as a single character or as a hexadecimal value. For example, if columns of data are separated by an ampersand, specify DELIMITER='&'.

If you omit DELIMITER=, the IMPORT procedure assumes that the delimiter is a space.

FMTLIB Statement

Saves value labels to the specified SAS format catalog.

Interaction: The FMTLIB statement is valid only when DBMS=JMP.

Syntax

FMTLIB=*<libref.>format-catalog*;

Required Argument

<libref.> format-catalog
specifies the format catalog where the value labels are saved.

GETNAMES Statement

Specifies whether the IMPORT procedure generates SAS variable names from the data values in the first row in the input file.

Default: YES

Restrictions: Valid only with the IMPORT procedure.

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

Interaction: The GETNAMES statement is valid only for delimited files.

Examples: "Example 1: Importing a Delimited File" on page 255
"Example 2: Importing a Specific Delimited File Using a Fileref" on page 259
"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 266

Syntax

GETNAMES=YES | NO;

Required Argument

YES | NO

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

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

Note: If a data value in the first row in the input file is read and it contains special characters that are not valid in a SAS name, such as a blank, then SAS converts the character to an underscore. For example, the variable name **`Occupancy Code`** would become the SAS variable name **`Occupancy_Code`**. Because SAS variable names cannot begin with a number, GETNAMES= prefixes an underscore to a variable name rather than replace the value's first character. For example, **`2014.CHANGES`** becomes **`_2014_CHANGES`**.

GUESSINGROWS Statement

Specifies the number of rows of the file to scan to determine the appropriate data type and length for the variables.

Default: 20

Restriction: This value should be greater than the value specified for DATAROW.

Interaction: The GUESSINGROWS statement is valid only for delimited files.

Syntax

GUESSINGROWS=*n* | MAX;

Required Arguments

n

indicates the number of rows the IMPORT procedure scans in the input file to determine the appropriate data type and length of variables. The range is 1 to 2147483647 (or MAX). The scan data process scans from row 1 to the number that is specified by the GUESSINGROWS option.

Note: You can use PROC REGISTRY to change the default row value in the SAS Registry.

MAX

can be specified instead of 2147483647. Specifying the maximum value could adversely affect performance.

Using the IMPORT Procedure

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's statements to do the following:

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

You can also use these same statements to change the default values.

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.

If you modify an informat, also modify the format for that same variable. The informat and format for a given variable 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 variable to avoid truncation when the data is displayed. For example, if a character variable is 400 characters long but has a format of **`$char50`**, then only the first 50 characters are shown when the data is displayed.

Note: 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 in *SAS Viya Statements: Reference*.

Note: EFI does not recognize multi-byte character sets.

CAUTION:
Sequential access is not allowed when you use EFI.

TIP **Sharing Delimited Files Across Hosts:** When a delimited file is read into SAS using the IMPORT procedure, each row must end with a host-specific, end-of-line delimiter. If you share delimited files that were created on one host with another host, the default end-of-line delimiters will not match. When this occurs, you must specify the new host's end-of-line delimiter for your files. On Linux the default end-of-row delimiter is Linefeed (LF). To read a file that is native to Windows, use a FILENAME statement with the TERMSTR=CRLF option. For more information, see the FILENAME statement in *SAS Viya Statements: Reference*.

PROC IMPORT uses the NLNUM informat instead of the COMMA informat. When you import a file that contains values such as 14,000.01 that have commas, the COMMA informat removes the commas and other non-numeric characters from the numerical values. Removing these characters can cause interpretation errors in the values. NLNUM prevents these errors by using the specified value of the LOCALE system option to interpret numerical values that have commas.

For example, to enter the numerical equivalent of fourteen thousand and one hundredth, a person specifying LOCALE=English_UnitedStates would enter 14,000.01. A person specifying LOCALE=French_France would enter 14.000,01. NLNUM interprets either input value correctly and writes the correct value based on the specified locale. If you read in 14.000,01 with NLNUM and LOCALE=French_France, store it in a data set, and then write it with NLNUM and LOCALE=English_UnitedStates, it is displayed as 14,000.01.

For more information, see:

- "COMMA Informat" in *SAS Viya Formats and Informats: Reference*
- "NLNUM Informat" in *SAS Viya National Language Support: Reference Guide*
- "Overview of Locale Concepts for NLS" in *SAS Viya National Language Support: Reference Guide*

Examples: IMPORT Procedure

Example 1: Importing a Delimited File

Features: PROC IMPORT statement options
DATAFILE=

DBMS=
GETNAMES=
OUT=
REPLACE

Other features: DELIMITER= statement
OPTIONS statement
PRINT procedure

Details

This example imports the following delimited external file and creates a temporary SAS data set named WORK.MYDATA:

```
Region&State&Month&Expenses&Revenue
Southern&GA&JAN2001&2000&8000
Southern&GA&FEB2001&1200&6000
Southern&FL&FEB2001&8500&11000
Northern&NY&FEB2001&3000&4000
Northern&NY&MAR2001&6000&5000
Southern&FL&MAR2001&9800&13500
Northern&MA&MAR2001&1500&1000
;
```

Program

```
options nodate ps=60 ls=80;

proc import datafile="/pathname/delimiter.txt"
            dbms=dlm
            out=mydata
            replace;

     delimiter='&';

     getnames=yes;

run;


proc print data=mydata;
run;
```

Program Description

Set your system options. The NODATE option suppresses the display of the date and time in the output. The LINESIZE= option specifies the output line length, and the PAGESIZE= option specifies the number of lines on an output page.

```
options nodate ps=60 ls=80;
```

Specify the input file. Specify that the input file is a delimited file. Replace the data set if it exists. Identify the output SAS data set.

```
proc import datafile="/pathname/delimiter.txt"
            dbms=dlm
```

```
              out=mydata
              replace;
```

Specify delimiter as an & (ampersand).

```
        delimiter='&';
```

Generate variable names from first row of data.

```
        getnames=yes;

   run;
```

Print out the output data set.

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

Log Examples

The SAS log displays information about the successful import. For this example, the IMPORT procedure generates a SAS DATA step, as shown in the log that follows. The log is divided into sections only for documentation appearances.

Log 11.1 *External File Imported to Create a SAS Data Set*

```
1          options nodate ps=60 ls=80;
 2          proc import datafile="/userid/pathname/delimiter.txt"
 3          dbms=dlm
 4          out=mydata
 5          replace;
 6          delimiter='&';
 7          getnames=yes;
 8          run;

 9           /*************************************************************
 9        ! ***
 10          *   PRODUCT:   SAS
 11          *   VERSION:   V.03.01
 12          *   CREATOR:   External File Interface
 13          *   DATE:      02Sep2016
 14          *   DESC:      Generated SAS Datastep Code
 15          *   TEMPLATE SOURCE:  (None Specified.)
 16          *************************************************************
 16       ! ***/
 17             data WORK.MYDATA    ;
 18             %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
 19             infile '/u/userid/pathname/delimiter.txt' delimiter =
 19       ! '&' MISSOVER DSD lrecl=32767 firstobs=2 ;
 20                informat Region $8. ;
 21                informat State $2. ;
 22                informat Month MONYY7. ;
 23                informat Expenses best32. ;
 24                informat Revenue best32. ;
 25                format Region $8. ;
 26                format State $2. ;
 27                format Month MONYY7. ;
 28                format Expenses best12. ;
 29                format Revenue best12. ;
 30             input
 31                         Region $
 32                         State $
 33                         Month
 34                         Expenses
 35                         Revenue
 36             ;
 37             if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR
 37        ! detection macro variable */
 38             run;
```

```
NOTE: The infile '/u/userid/pathname/delimiter.txt' is:
      Filename=/u/userid/pathname/delimiter.txt,
      Owner Name=userid,Group Name=unix_pubs,
      Access Permission=-rwx------,
      Last Modified=02Sep2016:11:22:03,
      File Size (bytes)=259

NOTE: 8 records were read from the infile
      '/u/userid/pathname/delimiter.txt'.
      The minimum record length was 1.
      The maximum record length was 30.
NOTE: The data set WORK.MYDATA has 8 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds

8 rows created in WORK.MYDATA from /u/userid/pathname/delimiter.txt.

NOTE: WORK.MYDATA data set was successfully created.
NOTE: The data set WORK.MYDATA has 8 observations and 5 variables.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.25 seconds
      cpu time            0.05 seconds

39         proc print data=mydata;
40         run;

NOTE: There were 8 observations read from the data set WORK.MYDATA.
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.04 seconds
      cpu time            0.03 seconds
```

Output Examples

Output 11.1 *Data Set Work.MyData*

The SAS System

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 2: Importing a Specific Delimited File Using a Fileref

Features: PROC IMPORT statement options

DATAFILE=
DBMS=
GETNAMES=
OUT=
REPLACE

Other features: FILENAME statement
PRINT procedure

Details

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

```
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
```

Program

```
filename stdata '/userid/pathname/state_data.txt' lrecl=100;


proc import datafile=stdata
        dbms=dlm
        out=states
        replace;

     delimiter=' ';
     getnames=yes;
run;

proc print data=states;
run;
```

Program Description

Specify a filename.

```
filename stdata '/userid/pathname/state_data.txt' lrecl=100;
```

Specify the input file. Specify that the input file is a delimited file. Replace the data set if it exists. Identify the output SAS data set.

```
proc import datafile=stdata
        dbms=dlm
        out=states
        replace;
```

Specify a blank value for the DELIMITER statement. Generate variable names from the first row of data with the GETNAMES statement.

```
     delimiter=' ';
     getnames=yes;
```

```
run;
```

Print out the data set.

```
proc print data=states;
run;
```

Log Examples

The SAS log displays information about the successful import. For this example, the IMPORT procedure generates a SAS DATA step, as shown in the log that follows. The log is divided into sections only for documentation appearances.

Log 11.2 *Importing a Specific Delimited File Using a Fileref*

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
55
56         filename stdata '/userid/pathname/state_data.txt' lrecl=100;
57         proc import datafile=stdata dbms=dlm
58         out=states replace;
59         delimiter=' ';
60         getnames=yes;
61         run;

62          /*****************************************************************
63          *   PRODUCT:   SAS
64          *   VERSION:   V.03.01
65          *   CREATOR:   External File Interface
66          *   DATE:      02Sep2016
67          *   DESC:      Generated SAS Datastep Code
68          *   TEMPLATE SOURCE:  (None Specified.)
69          *****************************************************************/
70             data WORK.STATES    ;
71             %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
72             infile STDATA delimiter = ' ' MISSOVER DSD  firstobs=2 ;
73                informat Region $7. ;
74                informat State $16. ;
75                informat Capital $11. ;
76                informat Bird $20. ;
77                format Region $7. ;
78                format State $16. ;
79                format Capital $11. ;
80                format Bird $20. ;
81             input
82                         Region $
83                         State $
84                         Capital $
85                         Bird $
86             ;
87             if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
88             run;
```

```
NOTE: The infile STDATA is:
      Filename=/userid/pathname/state_data.txt,
      Owner Name=userid,Group Name=unix_pubs,
      Access Permission=-rwx------,
      Last Modified=02Sep2016:14:05:22,
      File Size (bytes)=225

NOTE: 5 records were read from the infile STDATA.
      The minimum record length was 32.
      The maximum record length was 44.
NOTE: The data set WORK.STATES has 5 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds



5 rows created in WORK.STATES from STDATA.



NOTE: WORK.STATES data set was successfully created.
NOTE: The data set WORK.STATES has 5 observations and 4 variables.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.13 seconds
      cpu time            0.04 seconds


89
90         proc print data=states;
91         run;

NOTE: There were 5 observations read from the data set WORK.STATES.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.10 seconds
      cpu time            0.04 seconds


92
93         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```

Output Examples

Output 11.2 *Work.States Data Set*

The SAS System

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 3: Importing a Tab-Delimited File

Features: PROC IMPORT statement options
DATAFILE=
DATAROW=
DBMS=
OUT=
REPLACE

Other features: DELIMITER= statement
PRINT procedure

Details

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

Note: If you copy these lines of data into a file, make sure that the columns are separated by tabs. A simple copy and paste does not insert tabs between the columns.

Input Data 11.1 *Input*

```
Name     Gender   Age
Joyce    F        11
Thomas   M        11
Jane     F        12
Louise   F        12
James    M        12
John     M        12
Robert   M        12
Alice    F        13
Barbara  F        13
Jeffery  M        13
Carol    F        14
Judy     F        14
Alfred   M        14
Henry    M        14
Jenet    F        15
Mary     F        15
Ronald   M        15
William  M        15
Philip   M        16
```

Program

```
proc import datafile='/userid/pathname/Class.txt'
            out=class
            dbms=dlm
            replace;

       datarow=5;

     delimiter='09'x;
run;
```

```
proc print data=class;
run;
```

Program Description

Specify the input file. The GETNAMES= option defaults to 'yes'. Specify that the input file is a delimited file. Replace the data set if it exists. Specify the output data set.

```
proc import datafile='/userid/pathname/Class.txt'
            out=class
            dbms=dlm
            replace;
```

The first row read will be row 5 due to the DATAROW= option specification.

```
         datarow=5;
```

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**.

```
       delimiter='09'x;
run;
```

Print out the output data set.

```
proc print data=class;
run;
```

Log Examples

The SAS log displays information about the successful import. For this example, the IMPORT procedure generates a SAS DATA step, as shown in the log that follows. The log is divided into sections only for documentation appearances.

Log 11.3 *Importing a Tab-Delimited File*

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 55
 56         proc import datafile='/userid/pathname/Class.txt' out=class dbms=dlm replace;
 57         datarow=5;
 58         delimiter='09'x;
 59         run;

 60          /*****************************************************************
 61          *   PRODUCT:   SAS
 62          *   VERSION:   V.03.01
 63          *   CREATOR:   External File Interface
 64          *   DATE:      02Sep2016
 65          *   DESC:      Generated SAS Datastep Code
 66          *   TEMPLATE SOURCE:  (None Specified.)
 67          *****************************************************************/
 68             data WORK.CLASS    ;
 69             %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
 70             infile '/userid/pathname/Class.txt' delimiter='09'x MISSOVER DSD lrecl=32767
firstobs=5 ;
 71                informat "Name     Gender   Age"N $20. ;
 72                format "Name     Gender   Age"N $20. ;
 73             input
 74                         "Name     Gender   Age"N $
 75             ;
 76             if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
 77             run;

 NOTE: The infile '/userid/pathname/Class.txt' is:
       Filename=/userid/pathname/Class.txt,
       Owner Name=userid,Group Name=unix_pubs,
       Access Permission=-rwx------,
       Last Modified=02Sep2016:15:39:54,
       File Size (bytes)=439

 NOTE: 16 records were read from the infile '/userid/pathname/Class.txt'.
       The minimum record length was 20.
       The maximum record length was 20.
 NOTE: The data set WORK.CLASS has 16 observations and 1 variables.
 NOTE: DATA statement used (Total process time):
       real time           0.01 seconds
       cpu time            0.01 seconds

 16 rows created in WORK.CLASS from /userid/pathname/Class.txt.

 NOTE: WORK.CLASS data set was successfully created.
 NOTE: The data set WORK.CLASS has 16 observations and 1 variables.
 NOTE: PROCEDURE IMPORT used (Total process time):
       real time           0.13 seconds
       cpu time            0.05 seconds

 78
 79         proc print data=class;
 80         run;

 NOTE: There were 16 observations read from the data set WORK.CLASS.
 NOTE: PROCEDURE PRINT used (Total process time):
       real time           0.12 seconds
       cpu time            0.04 seconds

 81
 82         OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```

Output Example

Output 11.3 *Work.Class Data Set*

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

Example 4: Importing a Comma-Delimited File with a CSV Extension

Features: PROC IMPORT statement options
- DATAFILE=
- DBMS=
- GETNAMES=
- OUT=
- REPLACE

Other features: PRINT procedure

Details

This example imports the following comma-delimited file and creates a temporary SAS data set named Work.Shoes.

```
"Africa","Boot","Addis Ababa","12","$29,761","$191,821","$769"
"Asia","Boot","Bangkok","1","$1,996","$9,576","$80"
"Canada","Boot","Calgary","8","$17,720","$63,280","$472"
"Central America/Caribbean","Boot","Kingston","33","$102,372","$393,376","$4,454"
```

```
"Eastern Europe","Boot","Budapest","22","$74,102","$317,515","$3,341"
"Middle East","Boot","Al-Khobar","10","$15,062","$44,658","$765"
"Pacific","Boot","Auckland","12","$20,141","$97,919","$962"
"South America","Boot","Bogota","19","$15,312","$35,805","$1,229"
"United States","Boot","Chicago","16","$82,483","$305,061","$3,735"
"Western Europe","Boot","Copenhagen","2","$1,663","$4,657","$129"
```

Program

```
proc import datafile="/userid/pathname/shoes.csv"
        out=shoes
        dbms=csv
        replace;


     getnames=no;
run;

proc print data=work.shoes;
run;
```

Program Description

Specify the input data file. Replace the data set if it exists. Specify the output data set.

```
proc import datafile="/userid/pathname/shoes.csv"
        out=shoes
        dbms=csv
        replace;
```

Setting the GETNAMES= option to 'no' specifies that the variable names in record 1 are not used.

```
     getnames=no;
run;
```

Print the data set.

```
proc print data=work.shoes;
run;
```

Log Examples

The SAS log displays information about the successful import. For this example, the IMPORT procedure generates a SAS DATA step, as shown in the log that follows. The log is divided into sections only for documentation appearances.

Log 11.4 *Importing a Comma-Delimited File*

```
1          proc import datafile="/userid/pathname/shoes.txt" out=shoes dbms=csv replace;
 2             getnames=no;
 3          run;

 4           /*****************************************************************
 5           *   PRODUCT:   SAS
 6           *   VERSION:   V.03.01
 7           *   CREATOR:   External File Interface
 8           *   DATE:      02Sep2016
 9           *   DESC:      Generated SAS Datastep Code
 10          *   TEMPLATE SOURCE:  (None Specified.)
 11          *****************************************************************/
 12             data WORK.SHOES    ;
 13             %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
 14             infile '/userid/pathname/shoes.txt' delimiter = ',' MISSOVER DSD lrecl=32767 ;
 15                informat VAR1 $27. ;
 16                informat VAR2 $6. ;
 17                informat VAR3 $13. ;
 18                informat VAR4 $4. ;
 19                informat VAR5 $10. ;
 20                informat VAR6 $10. ;
 21                informat VAR7 $8. ;
 22                format VAR1 $27. ;
 23                format VAR2 $6. ;
 24                format VAR3 $13. ;
 25                format VAR4 $4. ;
 26                format VAR5 $10. ;
 27                format VAR6 $10. ;
 28                format VAR7 $8. ;
 29             input
 30                         VAR1 $
 31                         VAR2 $
 32                         VAR3 $
 33                         VAR4 $
 34                         VAR5 $
 35                         VAR6 $
 36                         VAR7 $
 37             ;
 38             if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
 39             run;
```

```
NOTE: The infile '/userid/pathname/shoes.txt' is:
      Filename=/userid/pathname/shoes.txt,
      Owner Name=userid,Group Name=unix_pubs,
      Access Permission=-rwx------,
      Last Modified=02Sep2016:15:52:41,
      File Size (bytes)=657

NOTE: 10 records were read from the infile '/userid/pathname/shoes.txt'.
      The minimum record length was 51.
      The maximum record length was 81.
NOTE: The data set WORK.SHOES has 10 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

10 rows created in WORK.SHOES from /userid/pathname/shoes.txt.

NOTE: WORK.SHOES data set was successfully created.
NOTE: The data set WORK.SHOES has 10 observations and 7 variables.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.12 seconds
      cpu time            0.04 seconds

40
41         proc print data=work.shoes;
42         run;

NOTE: There were 10 observations read from the data set WORK.SHOES.
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.07 seconds
      cpu time            0.04 seconds
103
106        OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```

Output Examples

Output 11.4 *Work.Shoes Data Set*

The SAS System

Obs	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7
1	Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769
2	Asia	Boot	Bangkok	1	$1,996	$9,576	$80
3	Canada	Boot	Calgary	8	$17,720	$63,280	$472
4	Central America/Caribbean	Boot	Kingston	33	$102,372	$393,376	$4,454
5	Eastern Europe	Boot	Budapest	22	$74,102	$317,515	$3,341
6	Middle East	Boot	Al-Khobar	10	$15,062	$44,658	$765
7	Pacific	Boot	Auckland	12	$20,141	$97,919	$962
8	South America	Boot	Bogota	19	$15,312	$35,805	$1,229
9	United States	Boot	Chicago	16	$82,483	$305,061	$3,735
10	Western Europe	Boot	Copenhagen	2	$1,663	$4,657	$129

Chapter 12

JAVAINFO Procedure

Overview: JAVAINFO Procedure

The JAVAINFO procedure conveys diagnostic information to the user about the Java environment that SAS is using. The diagnostic information can be used to confirm that the SAS Java environment has been configured correctly, and can be helpful when reporting problems to SAS technical support. Also, PROC JAVAINFO is often used to verify that the SAS Java environment is working correctly because PROC JAVAINFO uses Java to report its diagnostics.

Syntax: JAVAINFO Procedure

Restriction: This procedure is not supported by the CAS engine.

PROC JAVAINFO <*option(s)*>;

Statement	Task
PROC JAVAINFO	Display diagnostic information about the SAS Java environment

PROC JAVAINFO Statement

Displays diagnostic information about the SAS Java environment.

Syntax

PROC JAVAINFO <*option(s)*>;

Optional Arguments

ALL
lists all Java System properties currently set in the SAS Java environment.

CLASSPATHS
lists Java System properties that give details about classpaths that Java is using.

HELP
provides assistance in using the JAVAINFO procedure.

JREOPTIONS
lists Java System properties that are set when the JREOPTIONS configuration option is specified.

- When used in PROC JAVAINFO, JREOPTIONS lists the Java System properties that are set when Java is started.
- When used in PROC OPTIONS, JREOPTIONS lists the Java options that the JREOPTIONS SAS option contains when SAS is started.

Note: SAS.cfg is the configuration file specified during installation, but other configuration files can be specified.

OS
lists Java System properties that give details about the operating system that SAS is running under.

version
lists Java System properties the Java Runtime Environment (JRE) is using in SAS.

Chapter 13

OPTIONS Procedure

Overview: OPTIONS Procedure

The OPTIONS procedure lists the current settings of SAS system options in the SAS log.

SAS system options control how SAS formats output, handles files, processes data sets, interacts with the operating environment, and does other tasks that are not specific to a single SAS program or data set. You use the OPTIONS procedure to obtain information about an option or a group of options. Here is some of the information that the OPTIONS procedure provides:

- the current value of an option and how it was set
- a description of an option
- valid syntax for the option, valid option values, and the range of values
- where you can set the system option
- if the option can be restricted by your site administrator
- if the option has been restricted
- system options that belong to a system option group

- system options that are specific for an operating environment
- if an option value has been modified by the INSERT or APPEND system options

For additional information about SAS system options, see *SAS Viya System Options: Reference*.

Syntax: OPTIONS Procedure

PROC OPTIONS <*option(s)*>;

Statement	Task	Example
PROC OPTIONS	List the current system option settings to the SAS Log	Ex. 1, Ex. 2, Ex. 3, Ex. 4

PROC OPTIONS Statement

Lists the current settings of SAS system options in the SAS log.

Examples:

"Example 1: Producing the Short Form of the Options Listing" on page 284
"Example 2: Displaying the Setting of a Single Option" on page 285
"Example 3: Displaying Expanded Path Environment Variables" on page 286
"Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options" on page 288

Syntax

PROC OPTIONS <*option(s)*>;

Summary of Optional Arguments

LISTGROUPS
lists the system option groups as well as a description of each group.

Choose the format of the listing

DEFINE
displays the short description of the option, the option group, and the option type.

EXPAND
when displaying a character option, replaces an environment variable in the option value with the value of the environment variable. EXPAND is ignored if the option is a Boolean option, such as CENTER or NOCENTER, or if the value of the option is numeric.

HEXVALUE
displays system option character values as hexadecimal values.

LOGNUMBERFORMAT
displays numeric system option values using locale-specific punctuation.

LONG
: lists each system option on a separate line with a description.

NOEXPAND
: when displaying a path, displays the path using environment variable(s) and not the value of the environment variable(s). This is the default.

NOLOGNUMBERFORMAT
: displays numeric system option values without using punctuation, such as a comma or a period. This is the default.

SHORT
: specifies to display a compressed listing of options without descriptions.

VALUE
: displays the option's value and scope, as well as how the value was set.

Restrict the number of options displayed

GROUP=*group-name*

GROUP=*(group-name–1 ... group-name-n*)
: displays the options in one or more groups specified by *group-name*.

HOST
: displays only host options.

LISTINSERTAPPEND
: lists the system options whose value can be modified by the INSERT and APPEND system options.

LISTRESTRICT
: lists the system options that can be restricted by your site administrator.

NOHOST
: displays only portable options.

OPTION=*option-name*

OPTION=(*option-name-1 … option-name-n*)
: displays information about one or more system options.

RESTRICT
: displays system options that the site administrator has restricted from being updated.

Optional Arguments

DEFINE

displays the short description of the option, the option group, and the option type. SAS displays information about when the option can be set, whether an option can be restricted, and the valid values for the option.

Interaction This option is ignored when SHORT is specified.

Example "Example 2: Displaying the Setting of a Single Option" on page 285

EXPAND

when displaying a character option, replaces an environment variable in the option value with the value of the environment variable. EXPAND is ignored if the option is a Boolean option, such as CENTER or NOCENTER, or if the value of the option is numeric.

Tip By default, some option values are displayed with expanded variables. Other options require the EXPAND option in the PROC OPTIONS statement. Use the DEFINE option in the PROC OPTIONS statement to

determine whether an option value expands variables by default or if the EXPAND option is required. If the output from PROC OPTIONS DEFINE shows the following information, you must use the EXPAND option to expand variable values:

```
Expansion: Environment variables, within the option value,
           are not expanded
```

See "NOEXPAND" on page 277 option to view paths that display the environment variable

Example "Example 3: Displaying Expanded Path Environment Variables" on page 286

GROUP=*group-name*
GROUP=*(group-name–1 ... group-name-n*)
displays the options in one or more groups specified by *group-name*.

Requirement When you specify more than one group, enclose the group names in parenthesis and separate the group names by a space.

See "Displaying Information about System Option Groups" on page 281

HEXVALUE
displays system option character values as hexadecimal values.

HOST
displays only host options.

See "NOHOST" on page 277 option to display only portable options.

LISTINSERTAPPEND
lists the system options whose value can be modified by the INSERT and APPEND system options. The INSERT option specifies a value that is inserted as the first value of a system option value list. The APPEND option specifies a value that is appended as the last value of a system option value list. Use the LISTINERTAPPEND option to display which system options can have values inserted at the beginning or appended at the end of their value lists.

See "INSERT= System Option" in *SAS Viya System Options: Reference* and

"APPEND= System Option" in *SAS Viya System Options: Reference*

Example "Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options" on page 288

LISTGROUPS
lists the system option groups as well as a description of each group.

See "Displaying Information about System Option Groups" on page 281

LISTRESTRICT
lists the system options that can be restricted by your site administrator.

See "RESTRICT" on page 277 option to list options that have been restricted by the site administrator

LONG

lists each system option on a separate line with a description. This is the default. Alternatively, you can create a compressed listing without descriptions.

See "SHORT" on page 278 option to produce a compressed listing without descriptions

Example "Example 1: Producing the Short Form of the Options Listing" on page 284

LOGNUMBERFORMAT

displays numeric system option values using locale-specific punctuation.

See "NOLOGNUMBERFORMAT" on page 277 option to display numeric option values without using commas

Example "Example 2: Displaying the Setting of a Single Option" on page 285

NOEXPAND

when displaying a path, displays the path using environment variable(s) and not the value of the environment variable(s). This is the default.

See "EXPAND" on page 275 option to display a path by expanding the value of environment variables

NOHOST

displays only portable options.

Alias PORTABLE or PORT

See "HOST" on page 276 option to display only host options

NOLOGNUMBERFORMAT

displays numeric system option values without using punctuation, such as a comma or a period. This is the default.

See "LOGNUMBERFORMAT" on page 277 option to display numeric system options using commas

OPTION=*option-name*
OPTION=(*option-name-1 … option-name-n*)

displays a short description and the value (if any) of the option specified by *option-name*. DEFINE and VALUE options provide additional information about the option.

option-name

specifies the option to use as input to the procedure.

Requirement If a SAS system option uses an equal sign, such as PAGESIZE=, do not include the equal sign when specifying the option to OPTION=.

Example "Example 2: Displaying the Setting of a Single Option" on page 285

RESTRICT

displays the system options that have been set by your site administrator in a restricted options configuration file. These options cannot be changed by the user. For each option that is restricted, the RESTRICT option displays the option's value, scope, and how it was set.

If your site administrator has not restricted any options, then the following message appears in the SAS log:

```
Your Site Administrator has not restricted any SAS options.
```

See "LISTRESTRICT" on page 276 option to list options that can be restricted by the site administrator

SHORT
specifies to display a compressed listing of options without descriptions.

See "LONG" on page 277 option to create a listing with descriptions of the options.

VALUE
displays the option's value and scope, as well as how the value was set. If the value was set using a configuration file, the SAS log displays the name of the configuration file. If the option was set using the INSERT or APPEND system options, the SAS log displays the value that was inserted or appended.

Interaction This option has no effect when SHORT is specified.

Note SAS options that are passwords, such as METAPASS, return the value XXXXXXXX and not the actual password.

Example "Example 2: Displaying the Setting of a Single Option" on page 285

Displaying a List of System Options

The log that results from running PROC OPTIONS can show the system options for the options that are available for all operating environment and those that are specific to a single operating environment. Options that are available for all operating environments are referred to as portable options. Options that are specific to a single operating environment are referred to as host options.

The following example shows a partial log that displays the settings of session options. Your listing might differ.

```
proc options;
run;
```

Log 13.1 *The SAS Log Showing a Partial Listing of SAS System Options*

```
Portable Options:

 APPEND=           Specifies an option=value pair to insert the value at the end of the existing
                   option value.
 APPLETLOC=site-specific-path
                   Specifies the location of Java applets, which is typically a URL.
 AUTOCORRECT       Automatically corrects misspelled procedure names and keywords, and global
                   statement names.
 AUTOEXEC=/server-path/autoexec.sas
                    Specifies the location of the SAS AUTOEXEC files.
```

The log displays both portable and host options when you submit **`proc options;`**. The host options are specific for the Linux operating environment.

To view only host options, use this version of the OPTIONS procedure. Your listing might differ.

```
proc options host;
run;
```

Log 13.2 *The SAS Log Showing a Partial List of Host Options*

```
Host Options:

  ALIGNSASIOFILES   Aligns SAS files on a page boundary for improved performance.
  ALTLOG=           Specifies the location for a copy of the SAS log when SAS is running in batch mode.
  ALTPRINT=         Specifies the location for a copy of the SAS procedure output when SAS is running
in batch mode.
  BLKSIZE=256       Specifies the number of bytes that are read or written in one I/O operation.
```

Displaying Information about One or More Options

To view the setting of one or more particular options, you can use the OPTION= and DEFINE options in the PROC OPTIONS statement. The following example shows a log that PROC OPTIONS produces for a single SAS system option. Your output might differ.

```
proc options option=errorcheck define;
run;
```

Log 13.3 *The Setting of a Single SAS System Option*

```
 56         proc options option=errorcheck define; run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0

  ERRORCHECK=NORMAL
 Option Definition Information for SAS Option ERRORCHECK
     Group= ERRORHANDLING
     Group Description: Error messages and error conditions settings
     Description: Specifies whether SAS enters syntax-check mode when errors are found in the LIBNAME,
FILENAME,  %INCLUDE, and LOCK statements.
     Type: The option value is of type CHARACTER
           Maximum Number of Characters: 10
           Casing: The option value is retained uppercased
           Quotes: If present during "set", start and end quotes are removed
           Parentheses: The option value does not require enclosure within parentheses. If present,
the parentheses are retained.
           Expansion: Environment variables, within the option value, are not expanded
           Number of valid values: 2
                Valid value: NORMAL
                Valid value: STRICT
     When Can Set: Startup or anytime during the SAS Session
     Restricted: Your Site Administrator can restrict modification of this option
```

To view the settings for more than one option, enclose the options in parentheses and separate the options with a space:

```
proc options option=(append insert) define;
run;
```

Log 13.4 *The Settings of Two SAS System Options*

```
APPEND=
 Option Definition Information for SAS Option APPEND
     Group= ENVFILES
     Group Description: SAS library and file location information
     Description: Specifies an option=value pair to insert the value at the end of the existing option
value.
     Type: The option value is of type CHARACTER
           Maximum Number of Characters: 32000
           Casing: The option value is retained with original casing
           Quotes: If present during "set", start and end quotes are removed
           Parentheses: The option value does not require enclosure within parentheses. If present,
the parentheses are retained.
           Expansion: Environment variables, within the option value, are not expanded
     When Can Set: Startup or anytime during the SAS Session
     Restricted: Your Site Administrator cannot restrict modification of this option
```

```
INSERT=
 Option Definition Information for SAS Option INSERT
     Group= ENVFILES
     Group Description: SAS library and file location information
     Description: Specifies an option=value pair to insert the value at the beginning of the existing
option value.
     Type: The option value is of type CHARACTER
           Maximum Number of Characters: 32000
           Casing: The option value is retained with original casing
           Quotes: If present during "set", start and end quotes are removed
           Parentheses: The option value does not require enclosure within parentheses. If present,
the parentheses are retained.
           Expansion: Environment variables, within the option value, are not expanded
     When Can Set: Startup or anytime during the SAS Session
     Restricted: Your Site Administrator cannot restrict modification of this option
```

Displaying Information about System Option Groups

Each SAS system option belongs to one or more groups, which are based on functionality, such as error handling or sorting. You can display a list of system-option groups and the system options that belong to one or more of the groups.

Use the LISTGROUPS option to display a list of system-option groups. Your listing might differ.

```
proc options listgroups;
run;
```

Log 13.5 *List of SAS System Option Groups*

```
56         proc options listgroups; run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0


     Option Groups

        GROUP=CAS                    CAS Options

        GROUP=CODEGEN                Code generation

        GROUP=COMMUNICATIONS         Networking and encryption

        GROUP=DATACOM                Datacom

        GROUP=ENVFILES               Files

        GROUP=ERRORHANDLING          Error handling

        GROUP=EXECMODES              Initialization and operation

        GROUP=EXTFILES               External files

        GROUP=INPUTCONTROL           Data Processing

        GROUP=INSTALL                Installation
```

```
GROUP=LANGUAGECONTROL        Language control

GROUP=LISTCONTROL            Procedure output

GROUP=LOGCONTROL             SAS log

GROUP=LOG_LISTCONTROL        SAS log and procedure output

GROUP=MACRO                  SAS macro

GROUP=MEMORY                 Memory

GROUP=ODSPRINT               ODS Printing

GROUP=PDF                    PDF

GROUP=PERFORMANCE            Performance

GROUP=SASFILES               SAS Files

GROUP=SECURITY               Security

GROUP=SORT                   Procedure options

GROUP=SQL                    SQL
```

Use the GROUP= option to display system options that belong to a particular group. You can specify one or more groups.

```
proc options group=(cas memory);
run;
```

Log 13.6 *Sample Output Using the GROUP= Option*

```
 56         proc options group=(cas memory); run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0


 Group=CAS
  CASAUTHINFO=      Specifies an authinfo or netrc file that includes authentication information.
  CASHOST=cloud.sas.com
                    The CAS server name associated with a CAS session.
  CASLIB=           Specify the default CASLIB name.
  CASNAME=CASAUTO   Identify the name to associate with a generated CAS session.
  CASNWORKERS=ALL   Specify the number of workers to use with a CAS session.
  CASPORT=5570      The port associated with a CAS session.
  CASSESSOPTS=      Identify CAS server session options.
  CASTIMEOUT=60     The CAS session timeout in seconds.
  CASUSER=          The userid associated with a CAS session.

 Group=MEMORY
  SORTSIZE=1073741824
                    Specifies the amount of memory that is available to the SORT procedure.
  SUMSIZE=0         Specifies a limit on the amount of memory that is available for data summarization
procedures when class
                    variables are active.
  MAXMEMQUERY=268435456
                    For certain procedures, specifies the maximum amount of memory that can be
allocated per request.
  LOADMEMSIZE=0     Specifies a suggested amount of memory that is needed for executable programs
loaded by SAS.
  MEMSIZE=2147483648
                    Specifies the limit on the amount of virtual memory that can be used during a SAS
session.
  REALMEMSIZE=0     Specifies the amount of real memory SAS can expect to allocate.
```

You can use the following group names as values for the GROUP= option to list the system options in a group:

ANIMATION	EXECMODES	MEMORY
CAS	EXTFILES	ODSPRINT
CODEGEN	INPUTCONTROL	PDF
COMMUNICATIONS	INSTALL	PERFORMANCE
EMAIL	LANGUAGECONTROL	SASFILES
ENVDISPLAY	LOGCONTROL	SECURITY
ENVFILES	LOG_LISTCONTROL	SORT
ERRORHANDLING	MACRO	SQL

Displaying Restricted Options

Your site administrator can restrict some system options so that your SAS session conforms to options that are set for your site. Restricted options can be modified only by your site administrator. The OPTIONS procedure provides two options that display information about restricted options. The RESTRICT option lists the system options that your site administrator has restricted. The LISTRESTRICT option lists the options that can be restricted by your site administrator. For more information, see the listing of options that cannot be restricted .

The following SAS logs shows the output when the RESTRICT option is specified and partial output when the LISTRESTRICT option is specified. Your output might differ.

Log 13.7 *A List of Options That Have Been Restricted by the Site Administrator*

```
1
   proc options restrict;
2    run;
    SAS (r) Proprietary Software Release V.03.01  TS1M0

Option Value Information For SAS Option EMAILSYS
    Value: SMPT
    Scope: SAS Session
    How option value set:  Site Administrator Restricted
```

Log 13.8 *A Partial Log That Lists Options That Can Be Restricted*

```
56         proc options listrestrict; run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0


     Your Site Administrator can restrict the ability to modify the following Portable Options:

      APPLETLOC          Specifies the location of Java applets, which is typically a URL.
      AUTOCORRECT        Automatically corrects misspelled procedure names and keywords, and global
statement names.
      BINDING            Specifies the binding edge type of duplexed printed output.
      BUFNO              Specifies the number of buffers for processing SAS data sets.
      BUFSIZE            Specifies the size of a buffer page for output SAS data sets.
      BYERR              SAS issues an error message and stops processing if the SORT procedure
attempts to sort a _NULL_ data set.
      BYLINE             Prints the BY line above each BY group.
      BYSORTED           Requires observations in one or more data sets to be sorted in alphabetic or
numeric order.
```

Results: OPTIONS Procedure

SAS writes the options list to the SAS log. SAS system options of the form ***option*** | **NO*option*** are listed as either ***option*** or **NO*option***, depending on the current setting. They are always sorted by the positive form. For example, NOCAPS would be listed under the Cs.

The OPTIONS procedure displays passwords in the SAS log as eight Xs, regardless of the actual password length.

Examples: OPTIONS Procedure

Example 1: Producing the Short Form of the Options Listing

Features: PROC OPTIONS statement option

SHORT

Details

This example shows how to generate the short form of the listing of SAS system option settings. Compare this short form with the long form that is shown in "Displaying a List of System Options" on page 278.

Program

```
proc options short;
run;
```

Program Description

List all options and their settings. SHORT lists the SAS system options and their settings without any descriptions. Your output might differ.

```
proc options short;
run;
```

Log

Log 13.9 *Partial Listing of the SHORT Option*

```
56         proc options short; run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0


 Portable Options:

  APPEND= APPLETLOC=/pathname  AUTOCORRECT AUTOEXEC=/opt/sas/viya/etc/workspaceserver/autoexec.sas
 BINDING=DEFAULT BOTTOMMARGIN=0.000 IN BUFNO=1 BUFSIZE=0 BYERR BYLINE BYSORTED NOCAPS NOCARDIMAGE
CASAUTHINFO=  CASHOST=hostname  CASINSTALL= CASLIB= CASLIFETIME=1000000 CASLOGCNTL=NOMETRICS
CASNAME=CASAUTO CASNWORKERS=ALL
 CASPORT=7314 CASSESSOPTS= CASTIMEOUT=60 CASUSER= CATCACHE=0 CBUFNO=0 CENTER NOCHARCODE NOCHKPTCLEAN
```

Example 2: Displaying the Setting of a Single Option

Features: PROC OPTIONS statement option
- OPTION=
- DEFINE
- LOGNUMBERFORMAT
- VALUE

Details

This example shows how to display the setting of a single SAS system option. The log shows the current setting of the SAS system option MEMSIZE. The DEFINE and

VALUE options display additional information. The LOGNUMBERFORMAT displays the value using commas.

Program

```
proc options option=memsize define value lognumberformat;
run;
```

Program Description

Specify the MEMSIZE SAS system option. OPTION=MEMSIZE displays option value information. DEFINE and VALUE display additional information. LOGNUMBERFORMAT specifies to format the value using commas.

```
proc options option=memsize define value lognumberformat;
run;
```

Log

Log 13.10 *Log Output from Specifying the MEMSIZE Option*

```
56         proc options option=memsize define value lognumberformat;
 57         run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0

 Option Value Information For SAS Option MEMSIZE
     Value: 2,147,483,648
     Scope: SAS Session
     How option value set: Config File
     Config file name:
             /opt/sas/viya/SASFoundation/sasv9.cfg


 Option Definition Information for SAS Option MEMSIZE
     Group= MEMORY
     Group Description: Memory settings
     Group= PERFORMANCE
     Group Description: Performance settings
     Description: Specifies the limit on the amount of virtual memory that can be used during a SAS
session.
     Type: The option value is of type INTMAX
           Range of Values: The minimum is 0 and the maximum is 9223372036854775807
           Valid Syntax(any casing): MIN|MAX|n|nK|nM|nG|nT|hexadecimal
     Numeric Format: Usage of LOGNUMBERFORMAT impacts the value format
     When Can Set: Session startup (command line or config) only
     Restricted: Your Site Administrator can restrict modification of this option
```

Example 3: Displaying Expanded Path Environment Variables

Features: PROC OPTIONS statement options
- OPTION=
- EXPAND
- NOEXPAND
- HOST

Details

This example shows the value of an environment variable within an option value when the path is displayed.

Program

```
proc options option=msg expand;
run;
proc options option=msg noexpand;
run;
```

Program Description

Show the value of the environment variables within an option value: The EXPAND option causes the values of environment variables within the option value to display in place of the environment variable. The NOEXPAND option causes the environment variable within the options value to display. In this example, the environment variable is !sasroot

```
proc options option=msg expand;
run;
proc options option=msg noexpand;
run;
```

Log

Log 13.11 *Displaying an Expanded and Nonexpanded Pathname Using the OPTIONS Procedure*

```
56         proc options option=msg expand;
 57         run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0

  MSG=/opt/sas/viya/SASFoundation/sasmsg
                    Specifies the path to the library that contains SAS messages.
 NOTE: PROCEDURE OPTIONS used (Total process time):
       real time           0.00 seconds
       cpu time            0.00 seconds


 58         proc options option=msg noexpand;
 59         run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0

  MSG=!SASROOT/sasmsg
                    Specifies the path to the library that contains SAS messages.
```

Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options

Features: PROC OPTIONS statement option
LISTINSERTAPPEND

Details

This example shows how to display the options that can be specified by the INSERT and APPEND system options.

Program

```
proc options listinsertappend;
run;
```

Program Description

List all options that can be specified by the INSERT and APPEND options. The LISTINSERTAPPEND option provides a list and a description of these options. Your listing might differ.

```
proc options listinsertappend;
run;
```

Log

Log 13.12 *Displaying the Options That Can Be Specified by the INSERT and APPEND Options*

```
56         proc options listinsertappend; run;

     SAS (r) Proprietary Software Release V.03.01  TS1M0


     Core options that can utilize INSERT and APPEND

      AUTOEXEC           Specifies the location of the SAS AUTOEXEC files.
      FMTSEARCH          Specifies the order in which format catalogs are searched.
      SASAUTOS           Specifies the location of one or more autocall libraries.
      SASHELP            Specifies the location of the Sashelp library.
      SASSCRIPT          Specifies one or more locations of SAS/CONNECT server sign-on script files.

     Host options that can utilize INSERT and APPEND

      MSG                Specifies the path to the library that contains SAS messages.
      SET                Defines an environment variable.
```

Chapter 14

PRINT Procedure

Overview: PRINT Procedure

What Does the PRINT Procedure Do?

The PRINT procedure prints the rows in a SAS data set or rows from a Cloud Analytic Services (CAS) table using all or some of the variables. You can create a variety of reports ranging from printing all of the rows and columns of a table to a highly customized report that groups the data and calculates totals and subtotals for numeric columns.

A Simple Report

The following output illustrates the simplest type of report that you can produce. The statements that produce the output follow. "Example 2: Selecting Variables to Print" on page 314 creates the data set EXPREV.

```
options obs=10;

proc print data=exprev;
run;
```

The SAS System

Obs	Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
1	Antarctica	99999999	1/1/16	1/7/16	Internet	2	92.6	20.70
2	Puerto Rico	99999999	1/1/16	1/5/16	Catalog	14	51.2	12.10
3	Virgin Islands (U.S.)	99999999	1/1/16	1/4/16	In Store	25	31.1	15.65
4	Aruba	99999999	1/1/16	1/4/16	Catalog	30	123.7	59.00
5	Bahamas	99999999	1/1/16	1/4/16	Catalog	8	113.4	28.45
6	Bermuda	99999999	1/1/16	1/4/16	Catalog	7	41.0	9.25
7	Belize	120458	1/2/16	1/2/16	In Store	2	146.4	36.70
8	British Virgin Islands	99999999	1/2/16	1/5/16	Catalog	11	40.2	20.20
9	Canada	99999999	1/2/16	1/5/16	Catalog	100	11.8	5.00
10	Cayman Islands	120454	1/2/16	1/2/16	In Store	20	71.0	32.30

This next example creates the CAS table Mycas.Cars as a subset of the Sashelp.cars data set::

```
cas casauto;
libname mycas cas;

proc casutil outcaslib="casuser";
load data=sashelp.cars replace;
run;
```

```
data mycas.cars;
   set mycas.cars(where=(weight>6000));
   keep make model type;
run;

proc print data=mycas.cars;
   title "Cars Greater Than 6000 Pounds";
run;
```

Cars Greater Than 6000 Pounds

Obs	Make	Model	Type
1	Ford	Excursion 6.8 XLT	SUV
2	Hummer	H2	SUV
3	GMC	Yukon XL 2500 SLT	SUV

Customized Report

The following HTML5 report is a customized report that is produced by PROC PRINT. The statements that create this report do the following:

- customize the title and the column headings
- customize the appearance of the report
- place dollar signs and commas in numeric output
- selectively include and control the order of variables in the report
- group the data by JobCode
- sum the values for Salary for each job code and for all job codes, and add a label for the summary line and the grand total line

For an explanation of the program that produces this report, see “Program: Creating an HTML5 Report with the STYLE Option” on page 343.

Figure 14.1 *Customized Report Produced by PROC PRINT*

Expenses Incurred for
Salaries for Flight Attendants and Mechanics

Job Code	Gender	Annual Salary
FA1	F	$23,177.00
	F	$22,454.00
	M	$22,268.00
Total		$67,899.00

Job Code	Gender	Annual Salary
FA2	F	$28,888.00
	F	$27,787.00
	M	$28,572.00
Total		$85,247.00

Job Code	Gender	Annual Salary
FA3	F	$32,886.00
	F	$33,419.00
	M	$32,217.00
Total		$98,522.00

Job Code	Gender	Annual Salary
ME1	M	$29,769.00
Grand Total		$281,437.00

Concepts: PRINT Procedure

About PROC PRINT Output

By default, PROC PRINT produces an HTML5 report in SAS Viya. You can modify the appearance of the report by using PRINT procedure statements and options. The PRINT procedure statements, PROC PRINT, BY, PAGEBY, SUMBY, ID, SUM, and VAR control the content of the report. The options for each statement control the appearance of the report.

When you run your SAS programs in SAS Studio, you can check preferences in the **Results** tab to create PDF and RTF output when PROC PRINT executes. The HTML5, PDF, and RTF files are available for you to download from the program's **Results** tab.

When you run your SAS programs in batch or line modes and you want to create PDF or RTF output, you use ODS statements before the PROC PRINT statement. If you do not want HTML5 output, be sure to close the ODS HTML5 destination before you run the procedure:

```
ods html5 close;
```

See the PRINT procedure examples on page 314 for a sampling of the types of reports that the procedure produces.

Page Layout for HTML5, the Default ODS Destination

A page of ODS HTML5 output is not limited in width or length. Therefore, each row (observation) in a table is printed on a single line and all rows that are specified to print by the report appear on a single page of HTML5 output.

Each time PROC PRINT runs, by default, SAS adds a page break after the output. A page break is rendered by separating output with a horizontal rule.

Page Layout for PDF and RTF Destinations with Limited Page Sizes

Rows

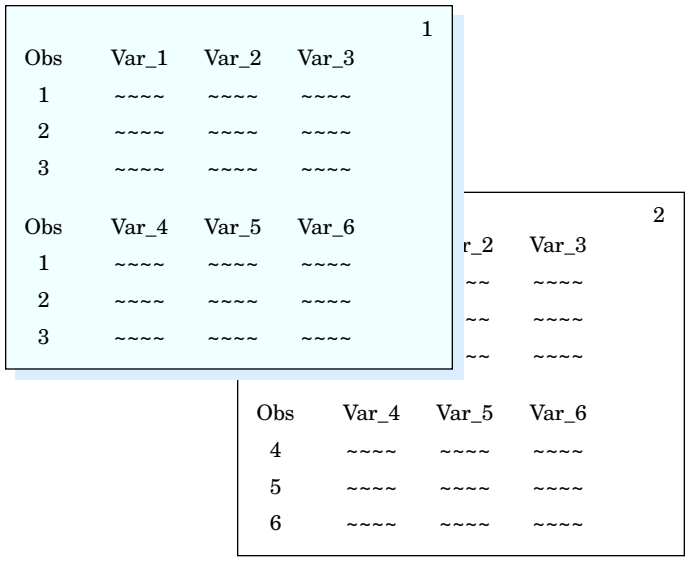
PROC PRINT uses an identical layout for all rows on a page for the PDF and RTF ODS destinations. These destinations produce output whose page size is limited in width and length. First, it attempts to print rows on a single line, as shown in the following figure.

Figure 14.2 *Printing Rows on a Single Line*

				1
Obs	Var_1	Var_2	Var_3	
1	~~~~	~~~~	~~~~	
2	~~~~	~~~~	~~~~	
3	~~~~	~~~~	~~~~	
4	~~~~	~~~~	~~~~	
5	~~~~	~~~~	~~~~	
6	~~~~	~~~~	~~~~	

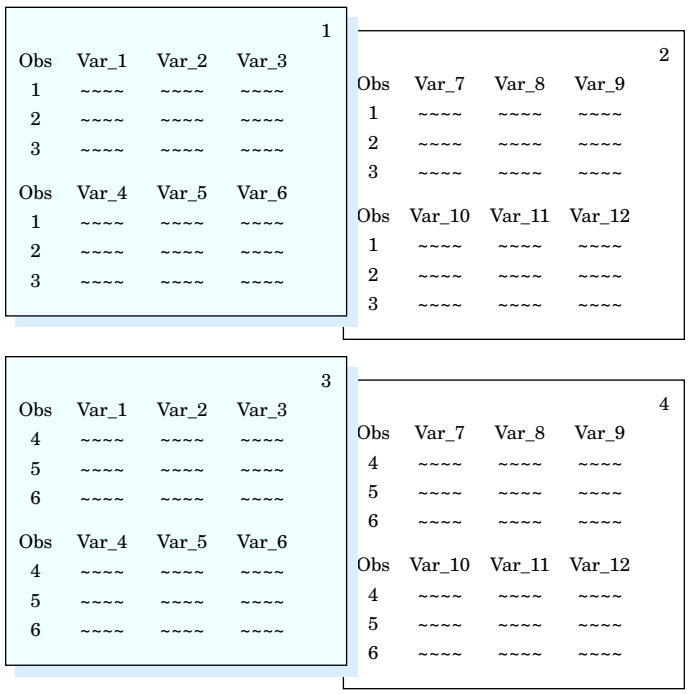
If PROC PRINT cannot fit all the variables on a single line, it splits the rows into two or more sections and prints the row number or the ID variables at the beginning of each line. For example, in the following figure, PROC PRINT prints the values for the first three variables in the first section of each page and the values for the second three variables in the second section of each page.

Figure 14.3 *Splitting Rows into Multiple Sections on One Page*



If PROC PRINT cannot fit all the variables on one page, the procedure prints subsequent pages with the same rows until it has printed all the variables. For example, in the following figure, PROC PRINT uses the first two pages to print values for the first three rows and the second two pages to print values for the rest of the rows.

Figure 14.4 *Splitting Rows across Multiple Pages*



Column Headings

The amount of spacing specifies whether PROC PRINT prints column headings horizontally or vertically. Figure 14.2 on page 293, Figure 14.3 on page 294, and Figure 14.4 on page 294 all illustrate horizontal headings. The following figure illustrates vertical headings.

Figure 14.5 *Using Vertical Headings*

```
                                        1
          V       V       V
          a       a       a
Obs       r       r       r
          –       –       –
          1       2       3

1       ~~~~    ~~~~    ~~~~
2       ~~~~    ~~~~    ~~~~
3       ~~~~    ~~~~    ~~~~
4       ~~~~    ~~~~    ~~~~
5       ~~~~    ~~~~    ~~~~
6       ~~~~    ~~~~    ~~~~
```

Note: If you use LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally unless you specify HEADING=VERTICAL.

Column Width

By default, PROC PRINT formats numeric data using the BEST. format and then uses the widest formatted value for the variable on the page as the column width.

If the formatted value of a character variable or the data width of an unformatted character variable exceeds the line size minus the length of all the ID variables, PROC PRINT might truncate the value. Consider the following situation:

- The line size is 80.
- IdNumber is a character variable with a length of 10. It is used as an ID variable.
- State is a character variable with a length of 2. It is used as an ID variable.
- Comment is a character variable with a length of 200.

When PROC PRINT prints these three variables on a line, it uses 14 print positions for the two ID variables, which includes a space after each ID variable. This arrangement leaves 80–14, or 66, print positions for COMMENT. Longer values of COMMENT are truncated.

Syntax: PRINT Procedure

Tip: You can use the ATTRIB, FORMAT, LABEL, TITLE, and WHERE statements. See *SAS Viya Statements: Reference*.

PROC PRINT <*option(s)*>;

BY <DESCENDING> *variable-1* <<DESCENDING> >*variable-2* … <NOTSORTED>;

PAGEBY *BY-variable*;

SUMBY *BY-variable*;

ID *variable(s)*
</ STYLE <(*location(s)*)>=<*style-override*>>;

SUM *variable(s)*
</ STYLE <(*location(s)*)>=<*style-override*>>;

VAR *variable(s)*

</ STYLE <(*location(s)*)>=<*style-override*> >;

Statement	Task	Example
PROC PRINT	Print rows in a data set	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 6, Ex. 8
BY	Produce a separate table in the report for each BY group	Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8
ID	Identify rows by the formatted values of the variables that you list instead of by row numbers	Ex. 8
PAGEBY	Control page ejects that occur before a page is full	Ex. 4
SUMBY	Limit the number of sums that appear in the report	Ex. 5, Ex. 6, Ex. 7, Ex. 8
SUM	Total values of numeric variables	Ex. 7
VAR	Select variables that appear in the report and determine their order	Ex. 2, Ex. 3, Ex. 8

PROC PRINT Statement

Prints rows in a SAS data set or CAS table using some or all of the variables.

Syntax

PROC PRINT <*option(s)*>;

Summary of Optional Arguments

CONTENTS=*link-text*
: specifies text for the links in the HTML5 contents file.

DATA=*SAS-data-set or CAS-table*
: specifies the SAS data set or the CAS table to print.

Control formatting

BLANKLINE=*n*

BLANKLINE=(COUNT=*n* <STYLE=[*style-attibute-specification(s)*]>)
: writes a blank line after *n* rows.

GRANDTOTAL_LABEL='*label*'
: displays a label on the grand total line.

HEADING=*direction*
: controls the orientation of the column headings.

LABEL
: specifies to use the variables' labels as column headings.

N<="*string-1*" <"*string-2*">>

prints the number of rows in the data set or CAS table in BY groups, or both and specifies explanatory text to print with the number.

NOOBS
suppresses the column in the output that identifies each row by number.

OBS="*column-header*"
specifies a column heading for the column that identifies each row by number.

ROUND
rounds unformatted numeric values to two decimal places.

SPLIT='*split-character*'
specifies the split character, which controls line breaks in column headings.

STYLE <(*locations(s)*)>=<*style-override(s)*>
specify one or more ODS style overrides to modify the default style element and attributes in a specific area of a report.

SUMLABEL

NOSUMLABEL

SUMLABEL='*label*'
specifies whether to display a label on the summary line for a BY group.

Optional Arguments

BLANKLINE=*n*
BLANKLINE=(COUNT=*n* <STYLE=[*style-attibute-specification(s)*]>)
specifies to insert a blank line after every *n* rows. The row count is reset to 0 at the beginning of each BY group.

n* | COUNT=*n
specifies the row number after which SAS inserts a blank line.

Requirement When you use the STYLE option, you must specify COUNT=*n* rather than specifying only *n*.

STYLE=[*style-attribute-specification(s)*]
specifies the style attribute to use for the blank line.

Default DATA

Tip You can use the BACKGROUNDCOLOR style attribute to make a visual distinction between rows using color.

See The STYLE= option on page 301 for valid style attributes.

Example "Example 2: Selecting Variables to Print" on page 314

CONTENTS=*link-text*
specifies the text for the links in the HTML5 contents file to the output produced by the PROC PRINT statement.

Restriction CONTENTS= does not affect the HTML5 body file. It affects only the HTML5 contents file.

DATA=*SAS-data-set or CAS-table*
specifies the SAS data set or the CAS table to print.

Note If you omit the DATA= option, the procedure uses the value of the SAS system option _LAST_=. The default of _LAST_= is the most recently created SAS data set or CAS table in the current SAS job or session.

Tip Each password and encryption key data set option that you specify must be coded on a separate line to ensure that they are properly blotted in the log.

GRANDTOTAL_LABEL='*label*'
displays a label on the grand total line. You can include the #BYVAR and #BYVAL variables in '*label*'.

Aliases GRAND_LABEL

GRANDTOT_LABEL

GTOT_LABEL

GTOTAL_LABEL

Example "Example 6: Summing Numeric Variables with Multiple BY Variables" on page 331

HEADING=*direction*
controls the orientation of the column headings, where *direction* is one of the following:

HORIZONTAL
prints all column headings horizontally.

Alias H

VERTICAL
prints all column headings vertically.

Alias V

Default Headings are either all horizontal or all vertical. If you omit HEADING=, PROC PRINT determines the direction of the column headings as follows:

If you do not use LABEL, spacing specifies whether column headings are vertical or horizontal.

If you use LABEL and at least one variable has a label, all headings are horizontal.

LABEL
specifies to use the variables' labels as column headings.

Alias L

Default PROC PRINT uses the name of the variable as the column heading in the following two circumstances:

1. if you omit the LABEL option in the PROC PRINT statement, even if the PROC PRINT step contains a LABEL statement
2. if a variable does not have a label

Interactions By default, if you specify LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally. Therefore,

using LABEL might increase the number of pages of output. (Use HEADING=VERTICAL in the PROC PRINT statement to print vertical column headings.)

PROC PRINT sometimes conserves space by splitting labels across multiple lines. Use SPLIT= in the PROC PRINT statement to control where these splits occur. You do not need to use LABEL if you use SPLIT=.

Note The LABEL system option must be in effect in order for any procedure to use labels. For more information see "LABEL System Option" in *SAS Viya System Options: Reference*.

Tip To create a blank column heading for a variable, use this LABEL statement in your PROC PRINT step:

```
label variable-name='00'x;
```

See For information about using the LABEL statement in a DATA step to create permanent labels, see "LABEL Statement" in *SAS Viya Statements: Reference*.

Example "Example 4: Creating Separate Sections of a Report for Groups of Rows Using a Batch Session" on page 321

N<="*string-1*" <"*string-2*">>

prints the number of rows in the data set or CAS table in BY groups, or both and specifies explanatory text to print with the number.

N Option Use	PROC PRINT Action
With neither a BY nor a SUM statement	Prints the number of rows in the data set at the end of the report and labels the number with the value of *string-1*.
With a BY statement	Prints the number of rows in the BY group at the end of each BY group and labels the number with the value of *string-1*.
With a BY statement and a SUM statement	Prints the number of rows in the BY group at the end of each BY group and prints the number of rows in the data set at the end of the report. The numbers for BY groups are labeled with *string-1*; the number for the entire data set is labeled with *string-2*.

Examples "Example 3: Customizing Text in Column Headings Using a Batch Session" on page 317

"Example 4: Creating Separate Sections of a Report for Groups of Rows Using a Batch Session" on page 321

"Example 5: Summing Numeric Variables with One BY Group" on page 328

NOOBS

suppresses the column in the output that identifies each row by number.

Example "Example 4: Creating Separate Sections of a Report for Groups of Rows Using a Batch Session" on page 321

OBS="*column-header*"

specifies a column heading for the column that identifies each row by number.

Tip OBS= honors the split character. (See the discussion of the SPLIT= option on page 300.)

Example "Example 3: Customizing Text in Column Headings Using a Batch Session" on page 317

ROUND

rounds unformatted numeric values to two decimal places. (Formatted values are already rounded by the format to the specified number of decimal places.) For both formatted and unformatted variables, PROC PRINT uses these rounded values to calculate any sums in the report.

If you omit ROUND, PROC PRINT adds the actual values of the rows to obtain the sum even though it displays the formatted (rounded) values. Any sums are also rounded by the format, but they include only one rounding error, that of rounding the sum of the actual values. The ROUND option, on the other hand, rounds values before summing them, so there might be multiple rounding errors. The results without ROUND are more accurate, but ROUND is useful for published reports where it is important for the total to be the sum of the printed (rounded) values.

Be aware that the results from PROC PRINT with the ROUND option might differ from the results of summing the same data with other methods such as the DATA step.

Alias R

CAUTION **Do not use ROUND with PICTURE formats.** ROUND is for use with numeric values. SAS procedures treat variables that have picture formats as character variables. Using ROUND with such variables might lead to unexpected results.

SPLIT='*split-character*'

specifies the split character, which controls line breaks in column headings. It also uses labels as column headings. PROC PRINT breaks a column heading when it reaches the split character and continues the header on the next line. The split character is not part of the column heading although each occurrence of the split character counts toward the 256-character maximum for a label.

Alias S=

Interactions You do not need to use both LABEL and SPLIT= because SPLIT= implies the use of labels.

The OBS= option honors the split character. (See the discussion of "OBS="*column-header*"" on page 300.)

Note PROC PRINT does not split labels of BY variables in the heading preceding each BY group, a summary label, or a grand total level,

even if you specify SPLIT=. Instead, PROC PRINT replaces the split character with a blank.

Example "Example 3: Customizing Text in Column Headings Using a Batch Session" on page 317

STYLE <(*locations(s)*)>=<*style-override(s)*>

specify one or more ODS style overrides to modify the default style element and attributes in a specific area of a report.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

style-element-name | [*style-attribute-name-1=style-attribute-value-1*
<*style-attribute-name-2=style-attribute-value-2* …>]

location

identifies the part of the report that the STYLE option affects. If *location(s)* is not specified, PROC PRINT determines the location to where the style override is applied based on the statement, the specified style element, and the style attribute.

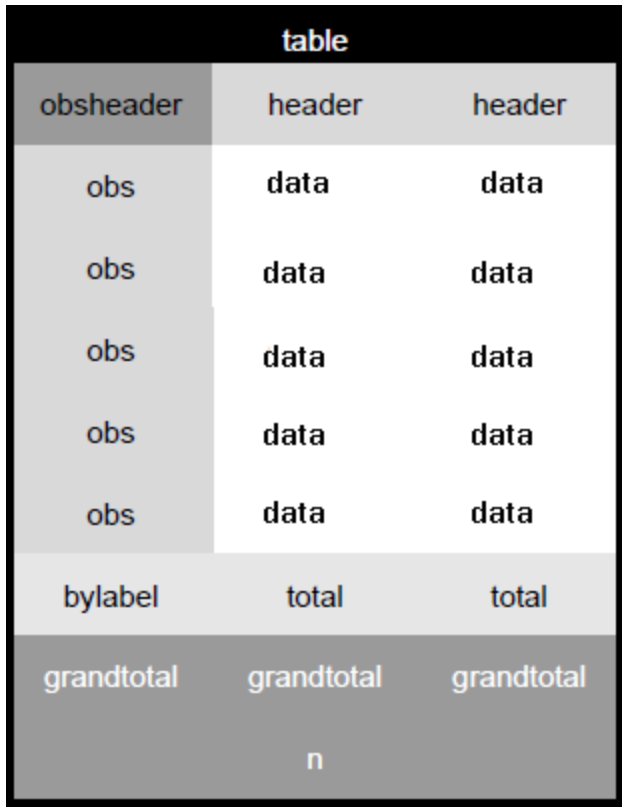
The following table shows the available locations and the other statements in which you can specify them.

Table 14.1 *Specifying Locations in the STYLE Option*

Location	Location Alias	Affected Report Part	Can Also Be Used in These Statements
BYLABEL	BYSUMLABEL BYLBL BYSUMLBL	Label for the BY variable on the line containing the SUM totals	None
DATA	COLUMN COL	All data except for data in the OBS column or the ID columns Or Data in the ID columns when the DATA location is specified in the STYLE= option of the ID statement	VAR ID SUM

Location	Location Alias	Affected Report Part	Can Also Be Used in These Statements
GRANDTOTAL	GRANDTOT GRAND GTOTAL GTOT	SUM line containing the grand totals for the whole report	SUM
HEADER	HEAD HDR	All column headings except for the OBS column or the ID columns Or All column headings of the ID columns when the HEADER location is specified in the STYLE= option of the ID statement	VAR ID SUM
N	None	N= table and contents	None
OBS	OBSDATA OBSCOLUMN OBSCOL	Data in the OBS column or the ID columns unless the DATA location is specified in the STYLE= option of the ID statement	None
OBSHEADER	OBSHEAD OBSHDR	Header of the OBS column or the ID columns unless the HEADER location is specified in the STYLE= option of the ID statement	None
TABLE	REPORT	Structural part of the report - that is, the underlying table used to set things like the width of the border and the space between cells	None
TOTAL	TOT BYSUMLINE BYLINE BYSUM	SUM line containing totals for each BY group	SUM

Figure 14.6 *PROC PRINT Areas and Corresponding Statements*



Style specifications in a statement other than the PROC PRINT statement override the same style specification in the PROC PRINT statement. However, style attributes that you specify in the PROC PRINT statement are inherited, provided that you do not override the style with style specifications in another statement. For example, if you specify a black background and a white foreground for all column headings in the PROC PRINT statement, and you specify a gray background for an ID column heading, the background for the ID column heading is gray, and the foreground is white (as specified in the PROC PRINT statement). This PRINT procedure of a CAS table shows the inheritance of the color white in the ID column heading:

```
proc print data=mycas.exprev style(head)={backgroundcolor=black color=white};
   id country / style(header data)=[backgroundcolor=gray];
run;
```

Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
Antarctica	99999999	1/1/16	1/7/16	Internet	2	92.6	20.70
Puerto Rico	99999999	1/1/16	1/5/16	Catalog	14	51.2	12.10

If the same style attributes appear for the OBSHEADER location in the PROC PRINT statement and the HEADER location in the ID statement, the HEADER location attributes override the OBSHEADER attributes. All other style attributes for the ID columns in both the PROC PRINT statement and the ID statement are merged to create the style for the ID columns. For example, in the PROC PRINT statement, the attributes for the OBSHEADER location are **`{fontsize=5 fontweight=bold}`**. In the ID statement, the attributes for the HEADER location are **`[fontsize=6 fontstyle=italic]`**. The resulting style for the ID column is **`[fontsize=6 fontweight=bold fontstyle=italic]`**.

```
proc print data=exprev style(obsheader)={fontsize=5 fontweight=bold};
   id country / style(header)=[fontsize=6 fontstyle=italic];
run;
```

Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
Antarctica	99999999	1/1/16	1/7/16	Internet	2	92.6	20.70

If the same style attributes appear for the OBS location in the PROC PRINT statement and the DATA location in the ID statement, the DATA location attributes override the OBS attributes. All other style attributes for the ID columns in both the PROC PRINT statement and the ID statement are merged to create the style for the ID columns. For example, in the PROC PRINT statement, the attributes for the OBS location are **{backgroundcolor=light gray color=blue}**. In the ID statement, the attributes for the DATA location are **[color=white fontstyle=italic]**. The resulting style for the ID column is **[backgroundcolor=light gray color=white fontstyle=italic]**.

```
proc print data=mycas.exprev style(obs)={backgroundcolor=light gray color=blue};
   id country / style(data)=[color=white fontstyle=italic];
run;
```

Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
Antarctica	99999999	1/1/16	1/7/16	Internet	2	92.6	20.70
Puerto Rico	99999999	1/1/16	1/5/16	Catalog	14	51.2	12.10
Virgin Islands (U.S.)	99999999	1/1/16	1/4/16	In Store	25	31.1	15.65
Aruba	99999999	1/1/16	1/4/16	Catalog	30	123.7	59.00

style-element-name
is the name of a style element. For a list of style elements, see "Style Elements for Use with ODS Graphics" in *SAS Viya ODS Graphics: Procedures Guide*.

When style elements are processed, more specific style elements override less specific style elements.

Tip You can use compound names and formats for style element names. An example of using a compound style element name is **style(obsheader)=data.italic.red;**. An example of using a format element name is **style=$cities**.

style-attribute-specification
describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

You can set these style attributes in the TABLE location:

BACKGROUNDCOLOR=	FONTWIDTH= *
BACKGROUNDIMAGE=	COLOR= *
BORDERCOLOR=	FRAME=
BORDERCOLORDARK=	HTMLCLASS=

BORDERCOLORLIGHT=	TEXTALIGN=
BORDERWIDTH=	OUTPUTWIDTH=
CELLPADDING=	POSTHTML=
CELLSPACING=	POSTIMAGE=
FONT= *	POSTTEXT=
FONTFAMILY= *	PREHTML=
FONTSIZE= *	PREIMAGE=
FONTSTYLE= *	PRETEXT=
FONTWEIGHT= *	RULES=

* When you use these attributes, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table.

You can set these style attributes in all locations other than TABLE:

ASIS=	FONTWIDTH=
BACKGROUNDCOLOR=	HREFTARGET=
BACKGROUNDIMAGE=	CLASS=
BORDERCOLOR=	TEXTALIGN=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
HEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=
FONT=	PRETEXT=
FONTFAMILY=	PROTECTSPECIALCHARACTERS=
FONTSIZE=	TAGATTR=
FONTSTYLE=	URL=

FONTWEIGHT=	VERTICALALIGN=

SUMLABEL
NOSUMLABEL
SUMLABEL='*label*'
specifies whether to display a label on the summary line for a BY group.

SUMLABEL
specifies to use the variable label, if it exists, as the label on the summary line in place of the variable name.

NOSUMLABEL
specifies leave the label on the summary line blank. Alternatively, you can use SUMLABEL="" (two single or double quotation marks with no space between them) to indicate a blank on the summary line.

SUMLABEL='*label*'
specifies the text to use as a label on the summary line of a BY group. You can include the #BYVAR and #BYVAL variables in '*label*'.

Default If you omit SUMLABEL, PROC PRINT uses the BY variable names in the summary line.

Examples "Example 5: Summing Numeric Variables with One BY Group" on page 328

"Example 6: Summing Numeric Variables with Multiple BY Variables" on page 331

BY Statement

Produces a separate section of the report for each BY group.

Examples:
"Example 4: Creating Separate Sections of a Report for Groups of Rows Using a Batch Session" on page 321
"Example 5: Summing Numeric Variables with One BY Group" on page 328
"Example 6: Summing Numeric Variables with Multiple BY Variables" on page 331
"Example 7: Limiting the Number of Sums in a Report" on page 336
"Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 340

Syntax

BY <DESCENDING> *variable-1* <<DESCENDING> >*variable-2* … <NOTSORTED>;

Required Argument

variable
specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, the rows in the data set must either be sorted by all the variables that you

specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING
specifies that the data set is sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED
specifies that rows are not necessarily sorted in alphabetic or numeric order. The data is grouped in another way, such as chronological order.

The requirement for ordering or indexing rows according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous rows that have the same values for all BY variables. If rows with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

Details

Using the BY Statement with an ID Statement

PROC PRINT uses a special layout if all BY variables appear in the same order at the beginning of the ID statement. (See "Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 340.)

Using the BY Statement with the NOBYLINE Option

If you use the BY statement with the SAS system option NOBYLINE, which suppresses the BY line that normally appears in output produced with BY-group processing, PROC PRINT always starts a new page for each BY group. This behavior ensures that if you create customized BY lines by putting BY-group information in the title and suppressing the default BY lines with NOBYLINE, the information in the titles matches the report on the pages.

Using a BY Variable When You Print Unsorted Data

If you specify a BY variable whose values are not sorted, SAS stops printing the data set when it processes the first unsorted group. A message is written to the SAS log.

ID Statement

Identifies rows by using the formatted values of the variables that you list instead of by using row numbers.

Example: "Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 340

Syntax

ID *variable(s)*
</ STYLE <(*location(s)*)>=<*style-override(s)*> >;

Required Argument

variable(s)

specifies one or more variables to print instead of the row number at the beginning of each row of the report.

Restriction If the ID variables occupy so much space that no room remains on the line for at least one other variable, PROC PRINT writes a warning to the SAS log and does not treat all ID variables as ID variables.

Interaction If a variable in the ID statement also appears in the VAR statement, the output contains two columns for that variable.

Optional Argument

STYLE <(*location(s)*)>=<*style-override(s)*>

specifies one or more style overrides to use for ID columns created with the ID statement.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

style-element-name | [*style-attribute-name-1=style-attribute-value-1* <*style-attribute-name-2=style-attribute-value-2* …>]

Restriction Style specifications for the OBSHEADER location is not valid in the ID statement.

Interaction If the STYLE(HEADER)= option is specified in the ID statement and the STYLE(OBSHEADER)= is specified in the PROC PRINT statement, the style attributes that are specified for the ID statement take precedence over the style elements that are specified in the PROC PRINT statement. Then, the style attributes in the PROC PRINT statement STYLE(OBSHEADER)= option are merged with the style attributes in the ID statement STYLE(HEADER)= option to render the output for the ID column heading.

Tip To specify different style overrides for different ID columns, use a separate ID statement for each variable and add a different STYLE option to each ID statement.

See For information about the arguments of this option and how it is used, see the STYLE= on page 301 option in the PROC PRINT statement.

Details

Using the BY Statement with an ID Statement

PROC PRINT uses a special layout if all BY variables appear in the same order at the beginning of the ID statement. (See “Example 8: Creating a Customized Layout with BY Groups and ID Variables” on page 340.)

PAGEBY Statement

Controls page ejects that occur before a page is full.

Requirement: BY statement

Example: "Example 4: Creating Separate Sections of a Report for Groups of Rows Using a Batch Session" on page 321

Syntax

PAGEBY *BY-variable*;

Required Argument

BY-variable
identifies a variable appearing in the BY statement in the PROC PRINT step. If the value of the BY variable changes, or if the value of any BY variable that precedes it in the BY statement changes, PROC PRINT begins printing a new page.

Interaction If you use the BY statement with the SAS system option NOBYLINE, which suppresses the BY line that normally appears in output produced with BY-group processing, PROC PRINT always starts a new page for each BY group. This behavior ensures that if you create customized BY lines by putting BY-group information in the title and suppressing the default BY lines with NOBYLINE, the information in the titles matches the report on the pages.

SUM Statement

Totals values of numeric variables.

Examples:
"Example 5: Summing Numeric Variables with One BY Group" on page 328
"Example 6: Summing Numeric Variables with Multiple BY Variables" on page 331
"Example 7: Limiting the Number of Sums in a Report" on page 336
"Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 340

Syntax

SUM *variable(s)*
</ STYLE <(*location(s)*)>=<*style-override(s)*> >;

Required Argument

variable(s)
identifies the numeric variables to total in the report.

Optional Argument

STYLE <(*location(s)*)>=<*style-override(s)*>
specifies one or more style overrides to use for cells containing sums that are created with the SUM statement.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

style-element-name | [*style-attribute-name-1=style-attribute-value-1* <*style-attribute-name-2=style-attribute-value-2* …>]

Tips To specify different style overrides for different cells reporting sums, use a separate SUM statement for each variable and add a different STYLE option to each SUM statement.

If the STYLE option is used in multiple SUM statements that affect the same location, the STYLE option in the last SUM statement will be used.

See For information about the arguments of this option and how it is used, see the option STYLE= on page 297 in the PROC PRINT statement.

Details

Using the SUM and BY Statements Together

When you use a SUM statement and a BY statement with one BY variable, PROC PRINT sums the SUM variables for each BY group that contains more than one row and totals them over all BY groups. (See "Example 5: Summing Numeric Variables with One BY Group" on page 328.)

When you use a SUM statement and a BY statement with multiple BY variables, PROC PRINT sums the SUM variables for each BY group that contains more than one row, just as it does if you use only one BY variable. However, it provides sums only for those BY variables whose values change when the BY group changes. (See "Example 6: Summing Numeric Variables with Multiple BY Variables" on page 331.)

Note: When the value of a BY variable changes, the SAS System considers that the values of all variables listed after it in the BY statement also change.

SUMBY Statement

Limits the number of sums that appear in the report.

Requirement: BY statement

Example: "Example 7: Limiting the Number of Sums in a Report" on page 336

Syntax

SUMBY *BY-variable*;

Required Argument

BY-variable
identifies a variable that appears in the BY statement in the PROC PRINT step. If the value of the BY variable changes, or if the value of any BY variable that precedes it in the BY statement changes, PROC PRINT prints the sums of all variables listed in the SUM statement.

Details

What Variables Are Summed?

If you use a SUM statement, PROC PRINT subtotals only the SUM variables. Otherwise, PROC PRINT subtotals all the numeric variables in the data set except for the variables listed in the ID and BY statements.

VAR Statement

Selects variables that appear in the report and determines their order.

Tip: If you omit the VAR statement, PROC PRINT prints all variables in the data set.

Examples: "Example 2: Selecting Variables to Print" on page 314
"Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 340

Syntax

VAR *variable(s)*
</ STYLE <(*location(s)*)>=<*style-override(s)*> >;

Required Argument

variable(s)
identifies the variables to print. PROC PRINT prints the variables in the order in which you list them.

Interaction In the PROC PRINT output, variables that are listed in the ID statement precede variables that are listed in the VAR statement. If a variable in the ID statement also appears in the VAR statement, the output contains two columns for that variable.

Optional Argument

STYLE <(*location(s)*)>=<*style-override(s)*>
specifies one or more style overrides to use for all columns that are created by a VAR statement.

You can specify a style override in two ways:

- Specify a style element. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program.
- Specify a style attribute. A style attribute is a name-value pair that describes a single behavioral or visual aspect of a piece of output. This is the most specific method of changing the appearance of your output.

style-override has the following form:

style-element-name | [*style-attribute-name-1=style-attribute-value-1* <*style-attribute-name-2=style-attribute-value-2* …>]

Tip To specify different style overrides for different columns, use a separate VAR statement to create a column for each variable and add a different STYLE option to each VAR statement.

See For information about the arguments of this option and how it is used, see the option STYLE= on page 301 in the PROC PRINT statement.

Printing CAS Tables

CAS is designed to process big data across multiple nodes. The row order for a table or a BY-group does not have significance for a CAS table. The order that CAS returns tables rows might be different each time you print a CAS table.

Error Processing in the PRINT Procedure Output

If an error occurs in the PRINT procedure or if the procedure is halted, output might be created for the rows that were processed until the error. SAS writes a message to the SAS log and ends the PRINT procedure.

Examples: PRINT Procedure

Example 1: Print a CAS Table

Features: PROC PRINT DATA=*CAS-table*

Other features: CAS language elements
CAS statement
LIBNAME statement for the CAS engine
PROC CASUTIL
PROC MDSUMMARY

Data set: Sashelp.cars

Details

This example demonstrates the following tasks:

- establishes a CAS session
- associates the Mycas libref with the CAS engine and the CAS session
- creates the CAS table mycas.cars
- uses PROC MDSUMMARY to summarize the cars data
- prints 15 rows of the summarized CAS table.

Program: Run in SAS Studio

```
cas mysess sessopts=(caslib='casuser');
libname mycas cas sessref=mysess;

proc casutil outcaslib="casuser";
  load data=sashelp.cars replace;
run;

proc mdsummary data=mycas.cars;
  var mpg_highway;
  groupby origin type / out=mycas.mpghw_sum;
run;


options obs=15;
proc print data=mycas.mpghw_sum;
   var origin type _mean_;
run;
```

Program Description

Set up the CAS session, create a libref for the CAS engine, and connect the engine to the CAS session. The CAS statement creates the Mysess session using the CASUSER caslib. The LIBNAME statement creates the Mycas libref for the CAS engine, which uses the Mysess CAS session.

```
cas mysess sessopts=(caslib='casuser');
libname mycas cas sessref=mysess;
```

Load the table Sashelp.cars into the caslib Casuser. The OUTCASLIB= option names the caslib to where the table is loaded. Use the LOAD statement to load the table from Sashelp.cars. The REPLACE option replaces the table in names the table to load

```
proc casutil outcaslib="casuser";
  load data=sashelp.cars replace;
run;
```

Summarize the data using PROC MDSUMMARY. The VAR statements specifies the analysis variable to order the results. The GROUPBY statement creates BY groups and saves the output to the table Mycas.mpghw_sum.

```
proc mdsummary data=mycas.cars;
  var mpg_highway;
  groupby origin type / out=mycas.mpghw_sum;
run;
```

Print the first 15 rows of the summary results. With OBS=15, PROC PRINT prints only 15 rows of the CAS table. The VAR statement limits the output table to three columns, Origin, Type, and _Mean_.

```
options obs=15;
proc print data=mycas.mpghw_sum;
   var origin type _mean_;
run;
```

Average Highway Milages

Obs	Origin	Type	_Mean_
1	Asia	Wagon	28.181818182
2	Europe	Sedan	27.115384615
3	Europe	Wagon	26.583333333
4	USA	SUV	20.04
5	USA	Wagon	29.714285714
6	Asia	Hybrid	56
7	Asia	SUV	21.68
8	USA	Sedan	28.544444444
9	USA	Truck	20.5
10	Asia	Truck	22
11	Europe	SUV	18.7
12	Asia	Sedan	29.968085106
13	Asia	Sports	26.647058824
14	Europe	Sports	25.130434783
15	USA	Sports	24.222222222

Example 2: Selecting Variables to Print

Features: PROC PRINT statement options
- BLANKLINE
- DOUBLE
- STYLE

VAR statement

Other features: FOOTNOTE statement

OPTIONS statement

TITLE statement

Data set: EXPREV

Details

This example demonstrates the following tasks:

- selects three variables for the reports
- uses variable labels as column headings

- creates a default HTML 5 report and a stylized HTML5 report

Program: Creating an HTML5 Report

```
options obs=10;

proc print data=exprev;

   var country price sale_type;
   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;
```

Program Description

HTML5 is the default destination.

Set the OBS= system option to process 10 rows.

```
options obs=10;
```

Print the output The VAR statement specifies the variables to print.

```
proc print data=exprev;

   var country price sale_type;
   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;
```

Output: HTML5

Output 14.1 *Selecting Variables: Default HTML5 Output*

Monthly Price Per Unit and Sale Type for Each Country

Obs	Country	Price	Sale_Type
1	Antarctica	92.6	Internet
2	Puerto Rico	51.2	Catalog
3	Virgin Islands (U.S.)	31.1	In Store
4	Aruba	123.7	Catalog
5	Bahamas	113.4	Catalog
6	Bermuda	41.0	Catalog
7	Belize	146.4	In Store
8	British Virgin Islands	40.2	Catalog
9	Canada	11.8	Catalog
10	Cayman Islands	71.0	In Store

*prices in USD

Program: Creating an HTML5 Report with the STYLE and BLANKLINE Options

```
options obs=5;


proc print data=exprev
   style(header)={fontstyle=italic color= green}
   style(obs)={backgroundcolor=#a8a44ff8a color=blue}
   blankline=(count= 1 style={backgroundcolor=cx456789});

   var country price sale_type;

   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;
```

Program Description

You can go a step further and add more formatting to your HTML5 output. The following example uses the STYLE option to add shading and spacing to your HTML5 report.

Process only the first five rows in the data set.

```
options obs=5;
```

Create stylized HTML5 output. The first STYLE option specifies that the column headings are written in green italic font. The second STYLE option specifies that row number column has a background color of the RGB color a8a44ff8a and a text color of blue. The BLANKLINE option specifies to add a blank line between each row and use a background color of the CMYK color cx456789. Because a style has not been defined for the OBSHEADER location, the Obs column heading in the output uses the default style color and not green.

```
proc print data=exprev
   style(header)={fontstyle=italic color= green}
   style(obs)={backgroundcolor=#a8a44ff8a color=blue}
   blankline=(count= 1 style={backgroundcolor=cx456789});

   var country price sale_type;

   title 'Monthly Price Per Unit and Sale Type for Each Country';
   footnote '*prices in USD';
run;
```

Output: HTML5 Output with Styles

Output 14.2 *Selecting Variables: HTML5 Output Using Styles*

Monthly Price Per Unit and Sale Type for Each Country

Obs	Country	Price	Sale_Type
1	Antarctica	92.6	Internet
2	Puerto Rico	51.2	Catalog
3	Virgin Islands (U.S.)	31.1	In Store
4	Aruba	123.7	Catalog
5	Bahamas	113.4	Catalog

*prices in USD

Example 3: Customizing Text in Column Headings Using a Batch Session

Features: PROC PRINT statement options
- N
- OBS=
- SPLIT=
- STYLE

VAR statement option
- STYLE

Other features: LABEL statement
ODS PDF statement
FORMAT statement
TITLE statement

Data set: EXPREV

ODS destination: PDF

Details

You can use this example in batch mode from either the Linux command line or SAS Studio Batch Submit. This example demonstrates the following tasks:

- specifies the ODS destination and the PDF file location and name
- adds background color to the column headings for variables in PDF output
- customizes the column heading for the column that identifies rows by number

- shows the number of rows in the report
- writes the values of the variable Price with dollar signs and periods

Program: Creating a PDF Report Using Batch

```
options obs=10;

ods pdf file='your_file.pdf';

proc print data=exprev label n obs='Observation Number';


   var country sale_type price;
         label country='Country Name'
               sale_type='Order Type'
               price='Price Per Unit in USD';
         format price dollar10.2;
         title 'Order Type and Price Per Unit in Each Country';
run;

ods pdf close;
```

Program Description

When you run SAS programs in batch mode either from the Linux command line or in SAS Studio, you can easily create PDF output by adding a few ODS statements. In the following example, ODS statements were added to produce PDF output. At the end of the program, the ODS PDF destination is closed.

The OBS= system option specifies to process 10 rows.

```
options obs=10;
```

Create PDF output and specify the file to store the output. The ODS PDF statement opens the PDF destination and creates PDF output. The FILE= argument specifies the external file that contains the PDF output.

```
ods pdf file='your_file.pdf';
```

Set the procedure options. The N option prints the number of rows at the end of the report. OBS= specifies the column heading for the column that identifies each row by number.

```
proc print data=exprev label n obs='Observation Number';
```

Process the variables in the data set. The VAR statement specifies the variables to print. The LABEL statement creates text to print in place of the variable names. The FORMAT statement specifies to format the price variables using the DOLLARw. format. The TITLE statement creates a title for the report.

```
   var country sale_type price;
         label country='Country Name'
               sale_type='Order Type'
               price='Price Per Unit in USD';
         format price dollar10.2;
         title 'Order Type and Price Per Unit in Each Country';
run;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

You can open the PDF from the file location that you specified in the ODS PDF statement.

Output 14.3 *Customizing Column Heading: Default PDF Output*

Order Type and Price Per Unit in Each Country

Observation Number	Country Name	Order Type	Price Per Unit in USD
1	Antarctica	Internet	$92.60
2	Puerto Rico	Catalog	$51.20
3	Virgin Islands (U.S.)	In Store	$31.10
4	Aruba	Catalog	$123.70
5	Bahamas	Catalog	$113.40
6	Bermuda	Catalog	$41.00
7	Belize	In Store	$146.40
8	British Virgin Islands	Catalog	$40.20
9	Canada	Catalog	$11.80
10	Cayman Islands	In Store	$71.00
N = 10			

Program: Creating a PDF Report with the STYLE Option, in Batch

```
options obs=10 nodate;

ods pdf file='your_file.pdf';

proc print data=exprev n split='*' obs='Observation Number'
     style(n)={backgroundcolor=light blue fontstyle=italic}
     style(header obs obsheader)={backgroundcolor=light yellow color=blue
      fontstyle=italic};


   var country sale_type price / style(data)=[backgroundcolor= light gray];
   label country='Country Name'
         sale_type='Order Type'
         price='Price Per Unit in USD';
   format price dollar10.2;
run;

title 'Order Type and Price Per Unit in Each Country';

ods pdf close;
```

Program Description

You can execute the following program in batch mode on the Linux command line or SAS Studio Batch Submit. If you execute this program using SAS Studio in non-interactive mode, you can omit the ODS statements if you have selected to created PDF output in the Preferences **Results** tab.

Set the OBS= system option to process 10 rows.

```
options obs=10 nodate;
```

Create PDF output and specify the file to store the output. The ODS PDF statement opens the PDF destination and creates PDF output. The FILE= argument specifies the external file that contains the PDF output.

```
ods pdf file='your_file.pdf';
```

Create stylized PDF output. The first STYLE option specifies that the background color of the cell containing the value for N be changed to light blue and that the font style be changed to italic. The second STYLE option specifies that the background color of the observation column, the observation header, and the other variable's headers be changed to a light yellow, the text color is changed to blue, and the font style is changed to italic.

```
proc print data=exprev n split='*' obs='Observation Number'
     style(n)={backgroundcolor=light blue fontstyle=italic}
     style(header obs obsheader)={backgroundcolor=light yellow color=blue
      fontstyle=italic};
```

Create stylized PDF output. The STYLE option changes the color of the cells containing data to a very light blue.

```
   var country sale_type price / style(data)=[backgroundcolor= light gray];
   label country='Country Name'
         sale_type='Order Type'
         price='Price Per Unit in USD';
   format price dollar10.2;
run;

title 'Order Type and Price Per Unit in Each Country';
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF Report with Styles

You can open the PDF from the file location that you specified in the ODS PDF statement.

Output 14.4 *Customizing Column Headings: PDF Using Styles*

Order Type and Price Per Unit in Each Country

Observation Number	Country Name	Order Type	Price Per Unit in USD
1	Antarctica	Internet	$92.60
2	Puerto Rico	Catalog	$51.20
3	Virgin Islands (U.S.)	In Store	$31.10
4	Aruba	Catalog	$123.70
5	Bahamas	Catalog	$113.40
6	Bermuda	Catalog	$41.00
7	Belize	In Store	$146.40
8	British Virgin Islands	Catalog	$40.20
9	Canada	Catalog	$11.80
10	Cayman Islands	In Store	$71.00
N = 10			

Example 4: Creating Separate Sections of a Report for Groups of Rows Using a Batch Session

Features: PROC PRINT statement options
- LABEL
- N=
- NOOBS
- STYLE

BY statement

PAGEBY statement

Other features: SORT procedure

FORMAT statement

LABEL statement

ODS RTF statement

TITLE statement

Data set: EXPREV

ODS destination: RTF

Details

These examples demonstrates the following:

- suppresses the printing of row numbers at the beginning of each row
- presents the data for each sale type in a separate section of the report

Note: By default, SAS Viya creates HTML5 output. You can set preferences in SAS Studio to create PDF and RTF output each time a program runs in the SAS Studio editor. You do not need to add ODS PDF and ODS RTF statements to your code. After execution, you can download the output file. This example shows the additional ODS statements to run programs in batch mode from the Linux command line or if you submit the program in SAS Studio using Batch Submit. If you run the example in the SAS Studio editor, remove the ODS PDF, ODS RTF, ODS PDF CLOSE, and ODS RTF CLOSE statements.

Program: Creating an RTF Report without Styles

```
options obs=10 nodate;

ods rtf file='your_file.rtf' startpage=no;

proc sort data=exprev;
   by sale_type order_date quantity;
run;

proc print data=exprev n='Number of observations for each order type:'
           noobs label;
   var quantity cost price;
   by sale_type order_date;
   pageby order_date;
   label sale_type='Order Type' order_date='Order Date';
   format price dollar7.2 cost dollar7.2;
   title 'Price and Cost Grouped by Date and Order Type';
   title2 'in USD';
run;

ods rtf close;
```

Program Description

The OBS= system option specifies to process ten rows.

```
options obs=10 nodate;
```

Create output for Microsoft Word and specify the file to store the output. The ODS RTF statement opens the RTF destination and creates output formatted for Microsoft Word. The FILE= option specifies the external file that contains the RTF output. The STARTPAGE=NO option specifies that no new pages be inserted explicitly at the start of each by group.

```
ods rtf file='your_file.rtf' startpage=no;
```

Sort the data. Use the BY statement in PROC SORT to sort the data using the variables Sale_type and Order_date for the report.

```
proc sort data=exprev;
   by sale_type order_date quantity;
run;

proc print data=exprev n='Number of observations for each order type:'
           noobs label;
   var quantity cost price;
```

```
   by sale_type order_date;
   pageby order_date;
   label sale_type='Order Type' order_date='Order Date';
   format price dollar7.2 cost dollar7.2;
   title 'Price and Cost Grouped by Date and Order Type';
   title2 'in USD';
run;
```

Close the RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Output 14.5 *Creating Separate Sections of a Report for Groups of Rows: Default RTF Output*

Price and Cost Grouped by Date and Order Type in USD

Order Type=Catalog Order Date=1/1/16

Quantity	Cost	Price
7	$9.25	$41.00
8	$28.45	$113.40
14	$12.10	$51.20
30	$59.00	$123.70
Number of observations for each order type:4		

Order Type=Catalog Order Date=1/2/16

Quantity	Cost	Price
11	$20.20	$40.20
100	$5.00	$11.80
Number of observations for each order type:2		

Order Type=In Store Order Date=1/1/16

Quantity	Cost	Price
25	$15.65	$31.10
Number of observations for each order type:1		

Order Type=In Store Order Date=1/2/16

Quantity	Cost	Price
2	$36.70	$146.40
20	$32.30	$71.00
Number of observations for each order type:2		

Order Type=Internet Order Date=1/1/16

Quantity	Cost	Price
2	$20.70	$92.60
Number of observations for each order type:1		

Program: Creating an RTF Report with the STYLE Option

```
options obs=10 nodate;

ods rtf file='your_file.rtf' startpage=no;

proc sort data=exprev;
   by sale_type order_date quantity;
run;

cas mysess sessopts=(caslib='casuser');
libname mycas cas '/u/userID/mycaslibs/' sessref=mysess;
proc casutil outcaslib="casuser";
   load data=exprev replace;
run;

proc print data=mycas.exprev n='Number of observations for the month: '
           noobs label style(N)={backgroundcolor=very light gray};
   var quantity / style(header)=[backgroundcolor=light yellow];
   var cost / style(header)=[backgroundcolor=blue foreground = white];
   var price / style(header)=[backgroundcolor=light green];
   by sale_type order_date;
   pageby order_date;
   label sale_type='Order Type' order_date='Order Date';
   format price dollar7.2 cost dollar7.2;

title 'Prices and Cost Grouped by Date and Order Type';
title2 '*prices in USD';
run;

ods rtf close;
```

Program Description

This program loads sorted data to SAS Viya and prints the CAS table with styles.

The OBS= system option specifies to process ten rows.

```
options obs=10 nodate;

ods rtf file='your_file.rtf' startpage=no;
```

Sort the data. Use the BY statement in PROC SORT to sort the data using the variables Sale_type and Order_date for the report.

```
proc sort data=exprev;
   by sale_type order_date quantity;
run;
```

Create a session with the SAS Viya server and load the Exprev data set to a CAS table. The CAS statement creates a session with the server using the Casuser caslib. The LIBNAME statement, using the CAS engine, connects the library to the SAS Viya session. PROC CASUTIL loads the Exprev data set to a CAS table in memory using the space designated for the Casuser caslib. When PROC PRINT executes, the CAS engine reads the data from the CAS table.

```
cas mysess sessopts=(caslib='casuser');
libname mycas cas '/u/userID/mycaslibs/' sessref=mysess;
proc casutil outcaslib="casuser";
   load data=exprev replace;
run;
```

Create a stylized RTF report. The first STYLE option specifies that the background color of the cell containing the number of rows be changed to light gray. The second STYLE option specifies that the background color of the column heading for the variable Quantity be changed to light yellow. The third STYLE option specifies that the background color of the column heading for the variable Cost be changed to light blue and the font color be changed to white. The fourth STYLE option specifies that the background color of the column heading for the variable Price be changed to light green.

```
proc print data=mycas.exprev n='Number of observations for the month: '
           noobs label style(N)={backgroundcolor=very light gray};
   var quantity / style(header)=[backgroundcolor=light yellow];
   var cost / style(header)=[backgroundcolor=blue foreground = white];
   var price / style(header)=[backgroundcolor=light green];
   by sale_type order_date;
   pageby order_date;
   label sale_type='Order Type' order_date='Order Date';
   format price dollar7.2 cost dollar7.2;

title 'Prices and Cost Grouped by Date and Order Type';
title2 '*prices in USD';
run;

ods rtf close;
```

Output: RTF with Styles

Output 14.6 *Creating Separate Sections of a Report for Groups of Rows: RTF Output Using Styles*

Prices and Cost Grouped by Date and Order Type
****prices in USD***

Order Type=Catalog Order Date=1/1/16

Quantity	Cost	Price
14	$12.10	$51.20
30	$59.00	$123.70
8	$28.45	$113.40
7	$9.25	$41.00
Number of observations for the month: 4		

Order Type=Catalog Order Date=1/2/16

Quantity	Cost	Price
100	$5.00	$11.80
11	$20.20	$40.20
Number of observations for the month: 2		

Order Type=In Store Order Date=1/1/16

Quantity	Cost	Price
25	$15.65	$31.10
Number of observations for the month: 1		

Order Type=In Store Order Date=1/2/16

Quantity	Cost	Price
20	$32.30	$71.00
2	$36.70	$146.40
Number of observations for the month: 2		

Order Type=Internet Order Date=1/1/16

Quantity	Cost	Price
2	$20.70	$92.60
Number of observations for the month: 1		

Example 5: Summing Numeric Variables with One BY Group

Features: PROC PRINT statement options
N=
SUMLABEL
BY statement
SUM statement

Other features: SORT procedure
TITLE statement
#BYVAL specification
SAS system options:
BYLINE
NOBYLINE

Data set: EXPREV

Details

This example demonstrates the following tasks:

- sums expenses and revenues for each region and for all regions.
- shows the number of rows in each BY group and in the whole report.
- creates a customized title, containing the name of the region. This title replaces the default BY line for each BY group.
- creates a default HTML5 file.

Program: Creating an HTML5 Report

```
options obs=10 nobyline;

proc sort data=exprev;
   by sale_type;
run;

cas mysess sessopts=(caslib='casuser');
libname mycas cas sessref=mysess;
proc casutil outcaslib="casuser";
   load data=exprev replace;
run;

proc print data=mycas.exprev noobs label sumlabel
           n='Number of observations for the order type: '
           'Number of observations for the data set: ';

   var country order_date quantity price;

   label  sale_type='Sale Type'
          price='Total Retail Price* in USD'
          country='Country' order_date='Date' quantity='Quantity';

   sum price quantity;
   by sale_type;

   format price dollar7.2;
```

```
   title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;

options byline;
```

Program Description

The HTML5 destination is open by default in SAS Studio. This program uses the default filename for the HTML5 output. You need the ODS HTML5 statement only for batch to specify the path and filename for the HTML5 output.

Start each BY group on a new page and suppress the printing of the default BY line. The SAS system option NOBYLINE suppresses the printing of the default BY line. When you use PROC PRINT with the NOBYLINE option, each BY group starts on a new page. The OBS= option specifies the number of rows to process.

```
options obs=10 nobyline;
```

Sort the data set. PROC SORT sorts the rows by Sale_Type.

```
proc sort data=exprev;
   by sale_type;
run;
```

Create a session with CAS and load the Exprev data set to a CAS table. The CAS statement creates a SAS Viya session using the Casuser caslib. The LIBNAME statement, using the CAS engine, connects the library Mycas to the SAS Viya session. PROC CASUTIL loads the Exprev data set to a CAS table in memory using the space designated for the Casuser caslib. When PROC PRINT executes, the CAS engine reads the data from the CAS table.

```
cas mysess sessopts=(caslib='casuser');
libname mycas cas sessref=mysess;
proc casutil outcaslib="casuser";
   load data=exprev replace;
run;
```

Print the report, suppress the printing of row numbers, and print the total number of rows for the selected variables. NOOBS suppresses the printing of row numbers at the beginning of the rows. SUMLABEL prints the BY variable label on the summary line of each. N= prints the number of rows in a BY group at the end of that BY group and (because of the SUM statement) prints the number of rows in the data set at the end of the report. The first piece of explanatory text that N= provides precedes the number for each BY group. The second piece of explanatory text that N= provides precedes the number for the entire data set.

```
proc print data=mycas.exprev noobs label sumlabel
           n='Number of observations for the order type: '
           'Number of observations for the data set: ';
```

Select the variables to include in the report. The VAR statement creates columns for Country, Order_Date, Quantity, and Price, in that order.

```
   var country order_date quantity price;
```

Assign the variables' labels as column headings. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step.

```
   label  sale_type='Sale Type'
```

```
            price='Total Retail Price* in USD'
            country='Country' order_date='Date' quantity='Quantity';
```

Sum the values for the selected variables. The SUM statement alone sums the values of Price and Quantity for the entire data set. Because the PROC PRINT step contains a BY statement, the SUM statement also sums the values of Price and Quantity for each sale type that contains more than one row.

```
    sum price quantity;
    by sale_type;
```

Format the numeric values for a specified column. The FORMAT statement assigns the DOLLAR7.2. format to Price for this report.

```
    format price dollar7.2;
```

Specify and format a dynamic (or current) title. The TITLE statement specifies a title. The #BYVAL specification places the current value of the BY variable Sale_Type in the title. Because NOBYLINE is in effect, each BY group starts on a new page, and the title serves as a BY line.

```
    title 'Retail and Quantity Totals for #byval(sale_type) Sales';
run;
```

Generate the default BY line. The SAS system option BYLINE resets the printing of the default BY line.

```
options byline;
```

Output: HTML5

Output 14.7 *Summing Numeric Variables with One BY Group HTML5 Output*

Retail and Quantity Totals for Catalog Sales

Country	Date	Quantity	Total Retail Price* in USD
Puerto Rico	1/1/16	14	$51.20
Aruba	1/1/16	30	$123.70
Bahamas	1/1/16	8	$113.40
Bermuda	1/1/16	7	$41.00
British Virgin Islands	1/2/16	11	$40.20
Canada	1/2/16	100	$11.80
Sale Type		**170**	**$381.30**
Number of observations for the order type: 6			

Retail and Quantity Totals for In Store Sales

Country	Date	Quantity	Total Retail Price* in USD
Virgin Islands (U.S.)	1/1/16	25	$31.10
Belize	1/2/16	2	$146.40
Cayman Islands	1/2/16	20	$71.00
Sale Type		**47**	**$248.50**
Number of observations for the order type: 3			

Retail and Quantity Totals for Internet Sales

Country	Date	Quantity	Total Retail Price* in USD
Antarctica	1/1/16	2	$92.60
		219	**$722.40**
Number of observations for the order type: 1 Number of observations for the data set: 10			

Example 6: Summing Numeric Variables with Multiple BY Variables

Features: PROC PRINT statement options
- GRANDTOTAL_LABEL=
- N=
- NOOBS
- STYLE
- SUMLABEL=

BY statement

SUM statement

Other features: ODS HTML5 statement

LABEL statement

FORMAT statement

SORT procedure

TITLE statement

Data set: EXPREV

ODS destination: HTML5

Details

This example demonstrates the following tasks:

- sums quantities and retail prices for the following items:
  - each order date

  - each sale type with more than one row in the report
  - all rows in the report
- shows the number of rows in each BY group and in the whole report
- displays a customized label in place of the BY group variable name on the summary line
- displays a customized label for the grand total line
- creates a default HTML5 report
- creates a stylized HTML5 report

Program: Creating an HTML5 Report

```
options obs=10;

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel='Totals' grandtotal_label='Grand Total';
   by sale_type order_date;
   sum price quantity cost;

   label  sale_type='Sale Type' order_date='Sale Date';
   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;
```

Program Description

```
options obs=10;
```

Produce HTML5 output by default. No ODS statement is necessary to create default HTML5 output. You can download the HTML5 file from the RESULTS tab.

```
proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel='Totals' grandtotal_label='Grand Total';
   by sale_type order_date;
   sum price quantity cost;

   label  sale_type='Sale Type' order_date='Sale Date';
   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;
```

Output: HTML5

Output 14.8 *Summing Numeric Variables with Multiple BY Variables: In Store Sales: Default HTML5 Output*

Retail and Quantity Totals for Each Sale Date and Sale Type

Sale Type=Catalog Sale Date=1/1/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/16	14	$51.20	$12.10
Aruba	99999999	1/4/16	30	$123.70	$59.00
Bahamas	99999999	1/4/16	8	$113.40	$28.45
Bermuda	99999999	1/4/16	7	$41.00	$9.25
Totals			**59**	**$329.30**	**$108.80**
N = 4					

Sale Type=Catalog Sale Date=1/2/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/16	11	$40.20	$20.20
Canada	99999999	1/5/16	100	$11.80	$5.00
Totals			**111**	**$52.00**	**$25.20**
Totals			**170**	**$381.30**	**$134.00**
N = 2					

Sale Type=In Store Sale Date=1/1/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/16	25	$31.10	$15.65
N = 1					

Sale Type=In Store Sale Date=1/2/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/16	2	$146.40	$36.70
Cayman Islands	120454	1/2/16	20	$71.00	$32.30
Totals			**22**	**$217.40**	**$69.00**
Totals			**47**	**$248.50**	**$84.65**
N = 2					

Sale Type=Internet Sale Date=1/1/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/16	2	$92.60	$20.70
Grand Total			**219**	**$722.40**	**$239.35**
N = 1 Total N = 10					

Program: Creating an HTML5 Report with the STYLE Option

```
options obs=10;

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel='Totals' grandtotal_label='Grand Total';

   by sale_type order_date;
sum price / style(GRANDTOTAL)=[backgroundcolor=white color=blue];
sum quantity / style(TOTAL)=[backgroundcolor=dark blue color=white];

   label  sale_type='Sale Type' order_date='Sale Date';
   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;
```

Program Description

```
options obs=10;

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev n noobs sumlabel='Totals' grandtotal_label='Grand Total';
```

Create stylized HTML5 output. The STYLE option in the first SUM statement specifies that the background color of the cell containing the grand total for the variable Price be changed to white and the font color be changed to blue. The STYLE option in the second SUM statement specifies that the background color of cells containing totals for the variable Quantity be changed to dark blue and the font color be changed to white.

```
   by sale_type order_date;
sum price / style(GRANDTOTAL)=[backgroundcolor=white color=blue];
sum quantity / style(TOTAL)=[backgroundcolor=dark blue color=white];

   label  sale_type='Sale Type' order_date='Sale Date';
   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Date and Sale Type';
run;
```

Output: HTML5 with Styles

Output 14.9 *Summing Numeric Variables with Multiple BY Variables: Catalog Sales: HTML5 Output Using Styles*

Retail and Quantity Totals for Each Sale Date and Sale Type

Sale Type=Catalog Sale Date=1/1/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/16	14	$51.20	$12.10
Aruba	99999999	1/4/16	30	$123.70	$59.00
Bahamas	99999999	1/4/16	8	$113.40	$28.45
Bermuda	99999999	1/4/16	7	$41.00	$9.25
Totals			**59**	**$329.30**	
N = 4					

Sale Type=Catalog Sale Date=1/2/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/16	11	$40.20	$20.20
Canada	99999999	1/5/16	100	$11.80	$5.00
Totals			**111**	**$52.00**	
Totals			**170**	**$381.30**	
N = 2					

Sale Type=In Store Sale Date=1/1/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/16	25	$31.10	$15.65
N = 1					

Sale Type=In Store Sale Date=1/2/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/16	2	$146.40	$36.70
Cayman Islands	120454	1/2/16	20	$71.00	$32.30
Totals			**22**	**$217.40**	
Totals			**47**	**$248.50**	
N = 2					

Sale Type=Internet Sale Date=1/1/16

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/16	2	$92.60	$20.70
Grand Total			**219**	**$722.40**	
N = 1 Total N = 10					

Example 7: Limiting the Number of Sums in a Report

Features: BY statement
SUM statement
SUMBY statement

Other features: FORMAT statement
LABEL statement
ODS PDF statement
SORT procedure
TITLE statement

Data set: EXPREV

Details

This example demonstrates the following tasks:

- creates a separate section of the report for each combination of sale type and sale date
- sums quantities and retail prices only for each sale type and for all sale types, not for individual dates
- displays PDF output

Program: Creating a PDF File

```
options obs=10;

proc sort data=exprev;
   by sale_type order_date;
run;
```

```
proc print data=exprev noobs sumlabel='Total' grandtotal_label='Grand Total';
   by sale_type order_date;
   sum price quantity;
   sumby sale_type;
   label sale_type='Sale Type' order_date='Sale Date';
   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Type';
run;
```

Program Description

The OBS= system option specifies to process ten rows.

```
options obs=10;
```

Sort the data set. PROC SORT sorts the rows by Sales_Type and Order_Date.

```
proc sort data=exprev;
   by sale_type order_date;
run;
```

Print the report and remove the observation numbers. NOOBS suppresses the printing of row numbers at the beginning of the rows. SUMLABEL uses the label for the BY variables on the summary line of each BY group. The totals are summed using the GRANDTOTAL_LABEL option which identifies the grand total using the label ‘Grand Total’.

```
proc print data=exprev noobs sumlabel='Total' grandtotal_label='Grand Total';
```

Sum the values for each region. The SUM and BY statements work together to sum the values of Price and Quantity for each BY group as well as for the whole report. The SUMBY statement limits the subtotals to one for each type of sale.

```
   by sale_type order_date;
   sum price quantity;
   sumby sale_type;
```

Assign labels to specific variables. The LABEL statement associates a label with the variables Sale_Type and Order_Date for the duration of the PROC PRINT step. These labels are used in the BY group title or the summary line.

```
   label sale_type='Sale Type' order_date='Sale Date';
```

Assign a format to the necessary variables and specify a title. The FORMAT statement assigns the COMMA10. format to Cost and Price for this report. The TITLE statement specifies a title.

```
   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Type';
run;
```

Output: PDF

To view the PDF, from the **RESULTS** tab click and then click **Open**.

Output 14.10 *Limiting the Number of Sums in a Report: PDF Output*

Retail and Quantity Totals for Each Sale Type

Sale Type=Catalog Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/12	14	$51.20	$12.10
Aruba	99999999	1/4/12	30	$123.70	$59.00
Bahamas	99999999	1/4/12	8	$113.40	$28.45
Bermuda	99999999	1/4/12	7	$41.00	$9.25

Sale Type=Catalog Sale Date=1/2/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/12	11	$40.20	$20.20
Canada	99999999	1/5/12	100	$11.80	$5.00
Total			170	$381.30	

Sale Type=In Store Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/12	25	$31.10	$15.65

Sale Type=In Store Sale Date=1/2/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/12	2	$146.40	$36.70
Cayman Islands	120454	1/2/12	20	$71.00	$32.30
Total			47	$248.50	

Sale Type=Internet Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/12	2	$92.60	$20.70
Grand Total			219	$722.40	

Program: Creating a PDF Report with the STYLE Option

```
options obs=10;

ods pdf file='your_file.pdf';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev noobs sumlabel='Total' grandtotal_label='Grand Total';

   by sale_type order_date;
```

```
   sum quantity price / style(TOTAL)=[backgroundcolor=light blue color=white];
   sum quantity price / style(GRANDTOTAL)=[backgroundcolor=green color=white];
   sumby sale_type;

   label sale_type='Sale Type' order_date='Sale Date';

   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Type';
run;

ods pdf close;
```

Program Description

```
options obs=10;

ods pdf file='your_file.pdf';

proc sort data=exprev;
   by sale_type order_date;
run;

proc print data=exprev noobs sumlabel='Total' grandtotal_label='Grand Total';

   by sale_type order_date;
```

Create stylized PDF output. The STYLE option in the first SUM statement specifies that the background color of cells containing totals for the variable Price be changed to light blue and the font color be changed to white. The STYLE option in the second SUM statement specifies that the background color of the cell containing the grand total for the Quantity variable be changed to yellow and the font color be changed to red.

```
   sum quantity price / style(TOTAL)=[backgroundcolor=light blue color=white];
   sum quantity price / style(GRANDTOTAL)=[backgroundcolor=green color=white];
   sumby sale_type;

   label sale_type='Sale Type' order_date='Sale Date';

   format price dollar10.2 cost dollar10.2;
   title 'Retail and Quantity Totals for Each Sale Type';
run;

ods pdf close;
```

Output: PDF with Styles

To view the PDF, from the **RESULTS** tab click and then click **Open**.

Output 14.11 *Limiting the Number of Sums in a Report: PostScript Output Using Styles*

Retail and Quantity Totals for Each Sale Type

Sale Type=Catalog Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Puerto Rico	99999999	1/5/12	14	$51.20	$12.10
Aruba	99999999	1/4/12	30	$123.70	$59.00
Bahamas	99999999	1/4/12	8	$113.40	$28.45
Bermuda	99999999	1/4/12	7	$41.00	$9.25

Sale Type=Catalog Sale Date=1/2/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
British Virgin Islands	99999999	1/5/12	11	$40.20	$20.20
Canada	99999999	1/5/12	100	$11.80	$5.00
Total			170	$381.30	

Sale Type=In Store Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Virgin Islands (U.S.)	99999999	1/4/12	25	$31.10	$15.65

Sale Type=In Store Sale Date=1/2/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Belize	120458	1/2/12	2	$146.40	$36.70
Cayman Islands	120454	1/2/12	20	$71.00	$32.30
Total			47	$248.50	

Sale Type=Internet Sale Date=1/1/12

Country	Emp_ID	Ship_Date	Quantity	Price	Cost
Antarctica	99999999	1/7/12	2	$92.60	$20.70
Grand Total			219	$722.40	

Example 8: Creating a Customized Layout with BY Groups and ID Variables

Features: PROC PRINT statement options
- SUMLABEL
- GRANDTOTAL_LABEL

BY statement

ID statement

SUM statement

VAR statement

Other features: SORT procedure

Data set: EMPDATA

Details

This customized report demonstrates the following tasks:

- selects variables to include in the report and the order in which they appear
- selects rows to include in the report
- groups the selected rows by JobCode
- sums the salaries for each job code and for all job codes
- displays numeric data with commas and dollar signs

Program: Creating an HTML5 Report

```
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;

proc print data=tempemp (obs=10) sumlabel='Total' grandtotal_label='Grand Total';

 id jobcode;
   by jobcode;
   var gender salary;

   sum salary;

   label jobcode='Job Code'
         gender='Gender'
         salary='Annual Salary';

   format salary dollar11.2;
   where jobcode contains 'FA' or jobcode contains 'ME';
   title 'Salary Expenses';
run;
```

Program Description

Create and sort a temporary data set. PROC SORT creates a temporary data set in which the rows are sorted by JobCode and Gender.

```
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;
```

Define the procedure options. The (obs=10) data set option sets the number of rows to process. The SUMLABEL option indicates to use the label 'Total' on the summary line for each BY group. The GRANDTOTAL_LABEL option indicates to use the label 'Grand Total' on the grand total line after all BY groups in the report.

```
proc print data=tempemp (obs=10) sumlabel='Total' grandtotal_label='Grand Total';
```

Specify the variables to include in the report. The VAR statement and the ID statement together select the variables to include in the report. The ID statement and the BY statement produce the special format.

```
  id jobcode;
    by jobcode;
    var gender salary;
```

Calculate the total value for each BY group. The SUM statement totals the values of Salary for each BY group and for the whole report.

```
    sum salary;
```

Assign labels to the appropriate variables. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step. When you use SPLIT= in the PROC PRINT statement, the procedure uses labels for column headings.

```
    label jobcode='Job Code'
          gender='Gender'
          salary='Annual Salary';
```

Create formatted columns. The FORMAT statement assigns a format to Salary for this report. The WHERE statement selects for the report only the rows for job codes that contain the letters 'FA' or 'ME'. The TITLE statement specifies the report title.

```
    format salary dollar11.2;
    where jobcode contains 'FA' or jobcode contains 'ME';
    title 'Salary Expenses';
run;
```

Output: HTML5

Output 14.12 *Creating a Customized Layout with BY Groups and ID Variables: Default HTML5 Output*

Salary Expenses

JobCode	Gender	Salary
FA1	F	$23,177.00
	F	$22,454.00
	M	$22,268.00
Total		$67,899.00

JobCode	Gender	Salary
FA2	F	$28,888.00
	F	$27,787.00
	M	$28,572.00
Total		$85,247.00

JobCode	Gender	Salary
FA3	F	$32,886.00
	F	$33,419.00
	M	$32,217.00
Total		$98,522.00

JobCode	Gender	Salary
ME1	M	$29,769.00
Grand Total		$281,437.00

Program: Creating an HTML5 Report with the STYLE Option

```
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;

proc print data=tempemp (obs=10)sumlabel='Total' grandtotal_label='Grand Total'
   style(HEADER)={fontstyle=italic}
   style(DATA)={backgroundcolor=blue foreground=white};

 id jobcode;
   by jobcode;
   var gender salary;
```

```
   sum salary  / style(total)={color=red};

   label jobcode='Job Code'
         gender='Gender'
         salary='Annual Salary';

   format salary dollar11.2;
   where jobcode contains 'FA' or jobcode contains 'ME';
   title 'Expenses Incurred for';
   title2 'Salaries for Flight Attendants and Mechanics';
run;
```

Program Description

```
proc sort data=empdata out=tempemp;
   by jobcode gender;
run;
```

Create stylized HTML5 output. The first STYLE option specifies that the font of the headers be changed to italic. The second STYLE option specifies that the background of cells that contain data be changed to blue and the foreground of these cells be changed to white. The SUMLABEL and GRANDTOTAL_LABEL options use a label in the summary and grand total lines, respectively, in place of variable names.

```
proc print data=tempemp (obs=10)sumlabel='Total' grandtotal_label='Grand Total'
   style(HEADER)={fontstyle=italic}
   style(DATA)={backgroundcolor=blue foreground=white};

 id jobcode;
   by jobcode;
   var gender salary;
```

Create total values that are written in red. The STYLE option specifies that the color of the foreground of the cell that contain the totals be changed to red.

```
   sum salary  / style(total)={color=red};

   label jobcode='Job Code'
         gender='Gender'
         salary='Annual Salary';

   format salary dollar11.2;
   where jobcode contains 'FA' or jobcode contains 'ME';
   title 'Expenses Incurred for';
   title2 'Salaries for Flight Attendants and Mechanics';
run;
```

Output: HTML5 with Styles

Output 14.13 *Creating a Customized Layout with BY Groups and ID Variables: HTML5 Output Using Styles*

Expenses Incurred for
Salaries for Flight Attendants and Mechanics

Job Code	Gender	Annual Salary
FA1	F	$23,177.00
	F	$22,454.00
	M	$22,268.00
Total		$67,899.00

Job Code	Gender	Annual Salary
FA2	F	$28,888.00
	F	$27,787.00
	M	$28,572.00
Total		$85,247.00

Job Code	Gender	Annual Salary
FA3	F	$32,886.00
	F	$33,419.00
	M	$32,217.00
Total		$98,522.00

Job Code	Gender	Annual Salary
ME1	M	$29,769.00
Grand Total		$281,437.00

Example 9: Printing All the Data Sets in a SAS Library

Features: Macro facility
DATASETS procedure
PRINT procedure

Data sets: PROCLIB.DELAY and
PROCLIB.INTERNAT from the Raw Data and DATA Steps appendix

ODS destination: HTML5

Details

This example prints all the data sets in a SAS library. You can use the same programming logic with any procedure. Just replace the PROC PRINT step near the end of the example with whatever procedure step you want to execute. The example uses the macro language. For details about the macro language, see *SAS Viya Macro Language: Reference*.

Program: Printing All of the Data Sets in a Library

```
libname printlib 'SAS-data-library';
libname proclib 'SAS-data-library';
options nodate pageno=1;

options obs=10;

proc datasets library=proclib memtype=data nolist;
   copy out=printlib;
      select delay internat;
run;

%macro printall(libname,worklib=work);

   %local num i;

   proc datasets library=&libname memtype=data nodetails;
      contents out=&worklib..temp1(keep=memname) data=_all_ noprint;
   run;

   data _null_;
      set &worklib..temp1 end=final;
      by memname notsorted;
      if last.memname;
      n+1;
      call symput('ds'||left(put(n,8.)),trim(memname));

      if final then call symput('num',put(n,8.));

   run;

   %do i=1 %to &num;
      proc print data=&libname..&&ds&i noobs;
         title "Data Set &libname..&&ds&i";
      run;
   %end;
%mend printall;

%printall(printlib)
```

Program Description

```
libname printlib 'SAS-data-library';
libname proclib 'SAS-data-library';
options nodate pageno=1;
```

Print ten rows.

```
options obs=10;
```

Copy the desired data sets from the WORK library to a permanent library. PROC DATASETS copies two data sets from the WORK library to the PRINTLIB library in order to limit the number of data sets available to the example.

```
proc datasets library=proclib memtype=data nolist;
   copy out=printlib;
      select delay internat;
run;
```

Create a macro and specify the parameters. The %MACRO statement creates the macro PRINTALL. When you call the macro, you can pass one or two parameters to it. The first parameter is the name of the library whose data set you want to print. The second parameter is a library used by the macro. If you do not specify this parameter, the WORK library is the default.

```
%macro printall(libname,worklib=work);
```

Create the local macro variables. The %LOCAL statement creates two local macro variables, NUM and I, to use in a loop.

```
%local num i;
```

Produce an output data set. This PROC DATASETS step reads the library that you specify as a parameter when you invoke the macro. The CONTENTS statement produces an output data set called TEMP1 in WORKLIB. This data set contains a row for each variable in each data set in the library LIBNAME. By default, each row includes the name of the data set that the variable is included in as well as other information about the variable. However, the KEEP= data set option writes only the name of the data set to TEMP1.

```
proc datasets library=&libname memtype=data nodetails;
   contents out=&worklib..temp1(keep=memname) data=_all_ noprint;
run;
```

Specify the unique values in the data set, assign a macro variable to each one, and assign DATA step information to a macro variable. This DATA step increments the value of N each time it reads the last occurrence of a data set name (when IF LAST.MEMNAME is true). The CALL SYMPUT statement uses the current value of N to create a macro variable for each unique value of MEMNAME in the data set TEMP1. The TRIM function removes extra blanks in the TITLE statement in the PROC PRINT step that follows.

```
data _null_;
   set &worklib..temp1 end=final;
   by memname notsorted;
   if last.memname;
   n+1;
   call symput('ds'||left(put(n,8.)),trim(memname));
```

Determine the number of rows in the DATA step. When it reads the last row in the data set (when FINAL is true), the DATA step assigns the value of N to the macro variable NUM. At this point in the program, the value of N is the number of rows in the data set.

```
   if final then call symput('num',put(n,8.));
```

Run the DATA step. The RUN statement is crucial. It forces the DATA step to run, thus creating the macro variables that are used in the CALL SYMPUT statements before the %DO loop, which uses them, executes.

```
run;
```

Print the data sets and end the macro. The %DO loop issues a PROC PRINT step for each data set. The %MEND statement ends the macro.

```
   %do i=1 %to &num;
      proc print data=&libname..&&ds&i noobs;
         title "Data Set &libname..&&ds&i";
      run;
   %end;
%mend printall;
```

Print all the data sets in the PRINTLIB library. This invocation of the PRINTALL macro prints all the data sets in the library PRINTLIB.

```
%printall(printlib)
```

Output: HTML5

Output 14.14 *Data Set PRINTLIB.DELAY*

Data Set printlib.DELAY

flight	date	orig	dest	delaycat	destype	delay
114	01MAR16	LGA	LAX	1-10 Minutes	Domestic	8
202	01MAR16	LGA	ORD	No Delay	Domestic	-5
219	01MAR16	LGA	LON	11+ Minutes	International	18
622	01MAR16	LGA	FRA	No Delay	International	-5
132	01MAR16	LGA	YYZ	11+ Minutes	International	14
271	01MAR16	LGA	PAR	1-10 Minutes	International	5
302	01MAR16	LGA	WAS	No Delay	Domestic	-2
114	02MAR16	LGA	LAX	No Delay	Domestic	0
202	02MAR16	LGA	ORD	1-10 Minutes	Domestic	5
219	02MAR16	LGA	LON	11+ Minutes	International	18

Output 14.15 *Data Set PRINTLIB.INTERNAT*

Data Set printlib.INTERNAT

flight	date	dest	boarded
219	01MAR16	LON	198
622	01MAR16	FRA	207
132	01MAR16	YYZ	115
271	01MAR16	PAR	138
219	02MAR16	LON	147
622	02MAR16	FRA	176
132	02MAR16	YYZ	106
271	02MAR16	PAR	172
219	03MAR16	LON	197
622	03MAR16	FRA	180

Chapter 15

PRINTTO Procedure

Overview: PRINTTO Procedure

The PRINTTO procedure defines locations, other than ODS destinations, for SAS procedure output and for the SAS log. By default, SAS procedure output and the SAS log are routed to the SAS Studio **RESULTS** tab and **LOG** tab, respectively.

Other than the SAS Studio locations, you can store the SAS log or procedure output in an external file or in a SAS catalog entry. Procedure output that is routed to a file or a catalog entry creates only monospace output. HTML, PDF, or RTF output cannot be created using the PRINTTO procedure.

Syntax: PRINTTO Procedure

PROC PRINTTO <*option(s)*>;

Statement	Task	Example
PROC PRINTTO	Define locations, other than ODS destinations, for SAS procedure output and for the SAS log	Ex. 1, Ex. 2

PROC PRINTTO Statement

Defines locations, other than ODS destinations, for SAS procedure output and for the SAS log.

Restrictions: The PRINTTO procedure does not define ODS destinations.The PRINTTO procedure can only route log messages to the log specified by the ALTLOG= system option when the code runs in Batch mode. The ALTLOG= option cannot be set in SAS Studio.

Tips: To reset the destination for the SAS log and procedure output to the default, use the PROC PRINTTO statement without options.

To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

Examples: "Example 1: Routing to External Files" on page 357

"Example 2: Routing to SAS Catalog Entries" on page 361

Syntax

PROC PRINTTO <*option(s)*>;

Summary of Optional Arguments

LABEL='*description*'
: provides a description for a SAS log or procedure output stored in a SAS catalog entry.

LOG=LOG | *file-specification* | *SAS-catalog-entry*
: routes the SAS log to a permanent external file or SAS catalog entry.

NEW
: replaces the file instead of appending to it.

PRINT= PRINT | *file-specification* | *SAS-catalog-entry*
: routes procedure output to a permanent external file or SAS catalog entry or printer.

UNIT=*nn*
: routes the output to the file identified by the fileref.

Without Arguments

When no options are specified, the PROC PRINTTO statement does the following:

- closes any files opened by a PROC PRINTTO statement
- points both the SAS log and SAS procedure output to their default locations

Interaction: To close the appropriate file and to return only the SAS log or procedure output to its default destination, use LOG=LOG or PRINT=PRINT.

Examples:

"Example 1: Routing to External Files" on page 357

"Example 2: Routing to SAS Catalog Entries" on page 361

Optional Arguments

LABEL='*description*'
provides a description for a catalog entry that contains a SAS log or procedure output.

Range 1–256 characters

Interaction Use the LABEL= option only when you specify a catalog entry as the value for the LOG= option or the PRINT= option.

Example "Example 2: Routing to SAS Catalog Entries" on page 361

LOG=LOG | *file-specification* | *SAS-catalog-entry*
routes the SAS log to one of three locations:

LOG
routes the SAS log to the SAS Studio LOG tab.

file-specification
routes the SAS log to an external file. *file-specification* can be one of the following:

'*external-file*'
the name of a fully qualified pathname that is specified in quotation marks.

Restriction *external-file* cannot be longer than 1024 characters.

log-filename
is an unquoted alphanumeric text string. SAS creates a log that uses *log-filename*.log as the log filename.

Requirements *log-filename* must be in the current directory.

Do not provide a file extension with *log-filename*.

environment-variable
specifies a UNIX environment variable that contains the pathname of the log file.

fileref
a fileref previously assigned to an external file.

SAS-catalog-entry
routes the SAS log to a SAS catalog entry. Express *SAS-catalog-entry* in one of the following ways:

***libref.catalog.entry*<.LOG>**
a SAS catalog entry stored in the SAS library and SAS catalog specified.

***catalog.entry*<.LOG>**
a SAS catalog entry stored in the specified SAS catalog in the default SAS library SASUSER.

***entry*.LOG**
a SAS catalog entry stored in the default SAS library and catalog: SASUSER.PROFILE.

Default By default, in Batch mode, *libref* is SASUSER, *catalog* is PROFILE, and *type* is OUTPUT. The Sasuser library is a Read-Only library in SAS Studio.

Default LOG

Interactions The SAS log and procedure output cannot be routed to the same catalog entry at the same time.

The NEW option replaces the existing contents of a file with the new log. Otherwise, the new log is appended to the file.

To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

When routing the log to a SAS catalog entry, you can use the LABEL option to provide a description for the entry in the catalog directory.

When the log is routed to a file other than the default log file and programs are submitted from multiple sources, the final SAS system messages that contain the real and CPU times are written to the default SAS log.

Tips After routing the log to an external file or a catalog entry, you can specify LOG to route the SAS log back to its default destination.

When routing the SAS log, include a RUN statement in the PROC PRINTTO statement. If you omit the RUN statement, the first line of the following DATA or PROC step is not routed to the new file. (This occurs because a statement does not execute until a step boundary is crossed.)

If you create a macro that contains a password and you do not want the password to appear in the SAS log, use the LOG=*file-specification* option to redirect the log to an external file.

When you specify LOG=, SAS stores the path of the SAS log file in the &SYSPRINTTOLOG automatic macro variable. You can use this macro variable to restore the previous SAS log file location. For more information, see "Restoring the Previous SAS Log or Output File Location" on page 356.

Examples "Example 1: Routing to External Files" on page 357

"Example 2: Routing to SAS Catalog Entries" on page 361

NEW

clears any information that exists in a file and prepares the file to receive the SAS log or procedure output.

Default If you omit NEW, the new information is appended to the existing file.

Interaction If you specify both LOG= and PRINT=, NEW applies to both.

Examples "Example 1: Routing to External Files" on page 357

"Example 2: Routing to SAS Catalog Entries" on page 361

PRINT= PRINT | *file-specification* | *SAS-catalog-entry*

routes procedure output to one of three locations:

PRINT
routes procedure output to the SAS Studio RESULTS tab.

Tip After routing it to an external file or a catalog entry, you can specify PRINT to route subsequent procedure output to its default destination.

file-specification
routes procedure output to an external file. *file-specification* can be one of the following:

'*external-file*'
the name of a fully qualified pathname that is specified in quotation marks.

Restriction *external-file* cannot be longer than 1024 characters.

print-filename
is an unquoted alphanumeric text string. SAS creates a print file that uses *print-filename* as the print filename.

Requirements *print-filename* must be in the current directory.

Do not provide a file extension with *print-filename*.

fileref
a fileref previously assigned to an external file.

Tip If you use the PRINTER fileref, output is sent directly to the printer.

environment-variable
specifies a UNIX environment variable that contains the pathname of the print file.

SAS-catalog-entry
routes procedure output to a SAS catalog entry. Express *SAS-catalog-entry* in one of the following ways:

***libref.catalog.entry*<.OUTPUT>**
a SAS catalog entry stored in the SAS library and SAS catalog specified.

***catalog.entry*<.OUTPUT>**
a SAS catalog entry stored in the specified SAS catalog in the default SAS library SASUSER.

***entry*.OUTPUT**
a SAS catalog entry stored in the default SAS library and catalog: SASUSER.PROFILE.

Default By default, in Batch mode, *libref* is SASUSER, *catalog* is PROFILE, and *type* is OUTPUT. The Sasuser library is a Read-Only library in SAS Studio.

Alias FILE=, NAME=

Default PRINT

Interactions The procedure output and the SAS log cannot be routed to the same catalog entry at the same time.

The NEW option replaces the existing contents of a file with the new procedure output. If you omit NEW, the new output is appended to the file.

To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

When routing procedure output to a SAS catalog entry, you can use the LABEL option to provide a description for the entry in the catalog directory.

Tip When you specify PRINT=, SAS stores the path of the output file in the &SYSPRINTTOLIST automatic macro variable. You can use this macro variable to restore the previous output file location. For more information, see "Restoring the Previous SAS Log or Output File Location" on page 356.

UNIT=*nn*
routes the output to the file identified by the fileref FT*nn*F001, where *nn* is an integer between 1 and 99.

Range 1–99, integer only.

Tips You can define this fileref yourself. However, some operating systems predefine certain filerefs in this form.

When you specify UNIT=, SAS stores the path of the output file in the &SYSPRINTTOLIST automatic macro variable. You can use this macro variable to restore the previous output file location. For more information, see "Restoring the Previous SAS Log or Output File Location" on page 356.

Setting Page Numbers Using SAS System Options

When the NUMBER SAS system option is in effect, there is a single page-numbering sequence for all output in the current job or session. When NONUMBER is in effect, output pages are not numbered.

You can specify the beginning page number for the output that you are currently producing by using the PAGENO= in an OPTIONS statement.

Restoring the Previous SAS Log or Output File Location

When you specify the LOG=, PRINT=, or the UNIT= options in the PROC PRINTTO statement, SAS stores the appropriate file location in automatic macro variables:

SYSPRINTTOLOG contains the path of the SAS log file location prior to redirection by the PRINTTO procedure

SYSPRINTTOLIST contains the path of the output file location prior to redirection by the PRINTTO procedure

To restore the previous file locations, you specify the appropriate automatic macro variable as the value of the LOG=, PRINT=, or UNIT= options. Here are some examples:

```
/* Restore the previous log and the output file locations. */
proc printto log=&sysprinttolog print=&sysprinttolist;
   run;

/* Restore the previous output file location. */
proc printto unit=&sysprinttolist;
   run;
```

To restore the log file or the procedure output file to SAS Studio, use LOG=LOG and PRINT=PRINT options, respectively.

Examples: PRINTTO Procedure

Example 1: Routing to External Files

Features: PRINTTO statement without options

PRINTTO statement options
- LOG=
- NEW
- PRINT=

Details

This example uses PROC PRINTTO to route the log and procedure output to an external file and then reset both locations to the default.

Program

```
options nodate pageno=1 linesize=80 pagesize=60 source;

proc printto log='log-file';
   run;

data numbers;
   input x y z;
   datalines;
 14.2   25.2   96.8
 10.8   51.6   96.8
  9.5   34.2  138.2
  8.8   27.6   83.2
 11.5   49.4  287.0
  6.3   42.0  170.7
;
```

```
proc printto print='output-file'
new;
run;

proc print data=numbers;
   title 'The Numbers Data Set';
run;

proc printto;
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. The SOURCE option writes lines of source code to the default destination for the SAS log.

```
options nodate pageno=1 linesize=80 pagesize=60 source;
```

Route the SAS log to an external file. PROC PRINTTO uses the LOG= option to route the SAS log to an external file. By default, this log is appended to the current contents of *log-file*.

```
proc printto log='log-file';
   run;
```

Create the NUMBERS data set. The DATA step uses list input to create the NUMBERS data set.

```
data numbers;
   input x y z;
   datalines;
 14.2   25.2   96.8
 10.8   51.6   96.8
  9.5   34.2  138.2
  8.8   27.6   83.2
 11.5   49.4  287.0
  6.3   42.0  170.7
;
```

Route the procedure output to an external file. PROC PRINTTO routes output to an external file. Because NEW is specified, any output written to *output-file* will overwrite the file's current contents.

```
proc printto print='output-file'
new;
run;
```

Print the NUMBERS data set. The PROC PRINT output is written to the specified external file.

```
proc print data=numbers;
   title 'The Numbers Data Set';
run;
```

Reset the SAS log and procedure output locations to default. PROC PRINTTO routes subsequent logs and procedure output to their default locations and closes both of the current files.

```
proc printto;
run;
```

Log

Log 15.1 *Portion of Log Routed to the Default Destination*

```
01   options nodate pageno=1 linesize=80 pagesize=60 source;
02
03   proc printto log='/u/home/mylogs/log1.log';
04   run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Log 15.2 *Portion of Log Routed to an External File*

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


61
62         data numbers;
63            input x y z;
64            datalines;

NOTE: The data set WORK.NUMBERS has 6 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


71         ;
72
73         proc printto print='/u/home/myOutput/print1.out' new;
74         run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.01 seconds
      cpu time            0.02 seconds


75
76         proc print data=numbers;
77            title 'The Numbers Data Set';
78         run;

NOTE: There were 6 observations read from the data set WORK.NUMBERS.
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.04 seconds


79
80         proc printto;
81         run;
```

Output

Output 15.1 *Procedure Output Routed to an External File*

The Numbers Data Set

Obs	x	y	z
1	14.2	25.2	96.8
2	10.8	51.6	96.8
3	9.5	34.2	138.2
4	8.8	27.6	83.2
5	11.5	49.4	287.0
6	6.3	42.0	170.7

Example 2: Routing to SAS Catalog Entries

Features: PRINTTO statement without options

PRINTTO statement options
- LABEL=
- LOG=
- NEW
- PRINT=

Details

This example uses PROC PRINTTO to route the SAS log and procedure output to a SAS catalog entry and then to reset both locations to the default.

Program

```
options source;

libname lib1 'SAS-library';

proc printto log='/u/home/myOutput/Inventory.log label='Inventory program' new;
run;

data lib1.inventry;
   length Dept $ 4 Item $ 6 Season $ 6 Year 4;
   input dept item season year @@;
   datalines;
3070 20410  spring 2015 3070 20411  spring 2016
3070 20412  spring 2016 3070 20413  spring 2016
3070 20414  spring 2014 3070 20416  spring 2015
3071 20500  spring 2013 3071 20501  spring 2016
3071 20502  spring 2015 3071 20503  spring 2016
3071 20505  spring 2014 3071 20506  spring 2015
3071 20507  spring 2016 3071 20424  spring 2016
;

proc printto print=lib1.cat1.inventry.output
             label='Inventory program' new;
run;


proc printto;
run;
```

Program Description

Set the SAS system options. The SOURCE option specifies to write source statements to the SAS log.

```
options source;
```

Assign a libref.

```
libname lib1 'SAS-library';
```

Route the SAS log to a SAS catalog entry. PROC PRINTTO routes the SAS log to a SAS catalog entry named SASUSER.PROFILE.TEST.LOG. The PRINTTO procedure uses the default libref and catalog SASUSER.PROFILE because only the entry name and type are specified. LABEL= assigns a description for the catalog entry.

```
proc printto log='/u/home/myOutput/Inventory.log label='Inventory program' new;
run;
```

Create the LIB1.INVENTORY data set. The DATA step creates a permanent SAS data set.

```
data lib1.inventry;
   length Dept $ 4 Item $ 6 Season $ 6 Year 4;
   input dept item season year @@;
   datalines;
3070 20410  spring 2015 3070 20411  spring 2016
3070 20412  spring 2016 3070 20413  spring 2016
3070 20414  spring 2014 3070 20416  spring 2015
3071 20500  spring 2013 3071 20501  spring 2016
3071 20502  spring 2015 3071 20503  spring 2016
3071 20505  spring 2014 3071 20506  spring 2015
3071 20507  spring 2016 3071 20424  spring 2016
;
```

Route the procedure output to a SAS catalog entry. LABEL= assigns a description for the catalog entry.

```
proc printto print=lib1.cat1.inventry.output
             label='Inventory program' new;
run;
```

Reset the SAS log and procedure output back to the default and close the file. PROC PRINTTO closes the current files that were opened by the previous PROC PRINTTO step and reroutes subsequent SAS logs and procedure output to their default locations.

```
proc printto;
run;
```

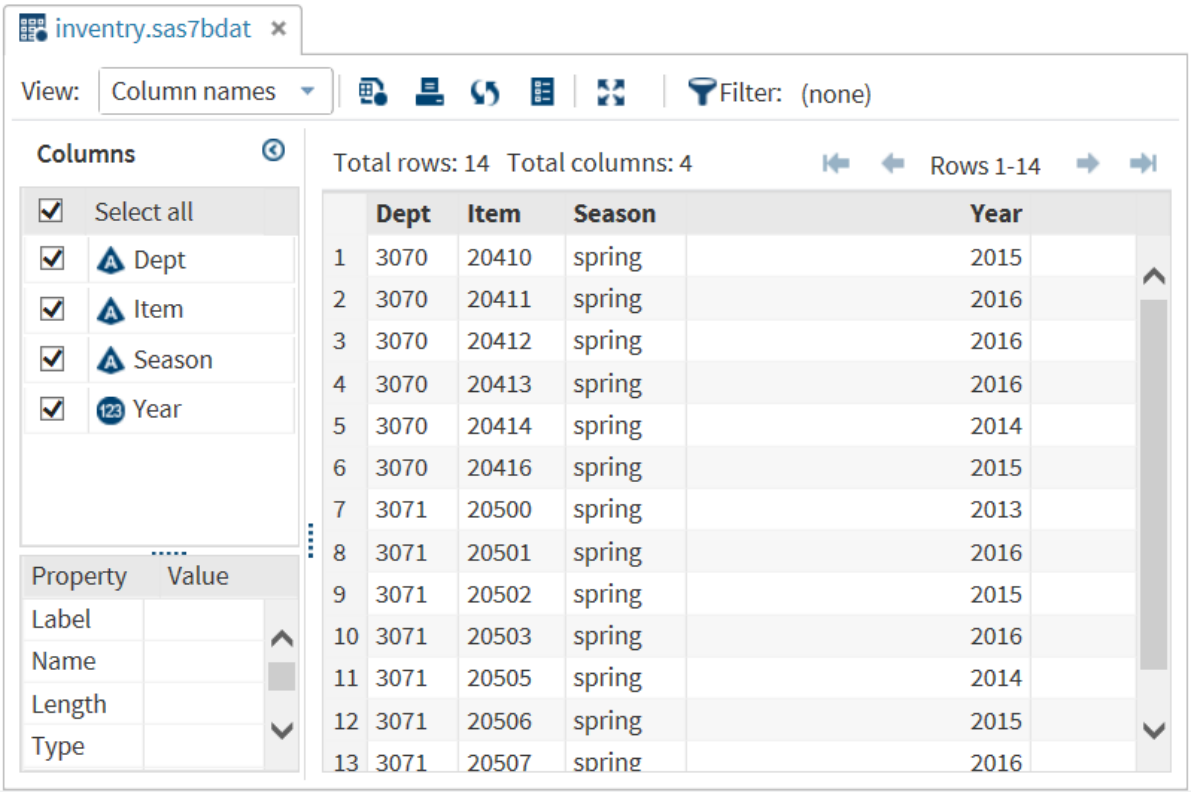
Log

Log 15.3 *SAS Log Routed to a SAS Catalog Entry*

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


62
63         data lib1.inventry;
64            length Dept $ 4 Item $ 6 Season $ 6 Year 4;
65            input dept item season year @@;
66            datalines;

NOTE: SAS went to a new line when INPUT statement reached past the end of a
      line.
NOTE: The data set LIB1.INVENTRY has 14 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.02 seconds


74         ;
75         proc printto print=lib1.cat1.inventry.output
76                      label='Inventory program' new;
77         run;

NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


78
79
80
81         proc printto;
82         run;
```

Output

Output 15.2 *Procedure Output Routed to SAS Catalog Entry LIB1.CAT1.INVENTRY.OUTPUT.*



	Dept	Item	Season	Year
1	3070	20410	spring	2015
2	3070	20411	spring	2016
3	3070	20412	spring	2016
4	3070	20413	spring	2016
5	3070	20414	spring	2014
6	3070	20416	spring	2015
7	3071	20500	spring	2013
8	3071	20501	spring	2016
9	3071	20502	spring	2015
10	3071	20503	spring	2016
11	3071	20505	spring	2014
12	3071	20506	spring	2015
13	3071	20507	spring	2016

Chapter 16

PRODUCT_STATUS Procedure

Overview: PRODUCT_STATUS Procedure

PROC PRODUCT_STATUS returns a list of the SAS Foundation products that are installed on your system, along with the version numbers of those products. It provides a quick method to determine whether a SAS product is available for your use. The results from PROC PRODUCT_STATUS are returned to the SAS log.

PROC PRODUCT_STATUS does not return information about web applications or other Java-based products.

The SYSVLONG and SYSVLONG4 automatic macro variables return only the version information for the SAS host image that is installed at your site. They do not return information for all of the SAS Foundation products that are installed at your site. For more information, see "SYSVLONG Automatic Macro Variable" in *SAS Viya Macro Language: Reference* and "SYSVLONG4 Automatic Macro Variable" in *SAS Viya Macro Language: Reference*.

Syntax: PRODUCT_STATUS Procedure

Restriction: This procedure is not supported by the CAS engine.

PROC PRODUCT_STATUS;

Statement	Task
"Example: Results from PROC PRODUCT_STATUS"	Specify the names and versions of the SAS Foundation products that are installed on your operating system.

PRODUCT_STATUS Statement

Returns the names and versions of the SAS Foundation products that are installed on your operating system.

Syntax

PROC PRODUCT_STATUS;

Details

The PROC PRODUCT STATUS statement does not have any arguments.

Example: Results from PROC PRODUCT_STATUS

```
proc product_status;
run;
```

Here is a partial output that contains an example of the results that are produced by PROC PRODUCT_STATUS.

```
For Base SAS Software ...
   Custom version information: V.03.01
   Image version information: V.03.01M0P082816
For SAS/CONNECT ...
   Custom version information: V.03.00
For CAS Action Set for Aggregation ...
   Custom version information: 1.0
For Action Set for Boolean Rule Extraction ...
   Custom version information: 1.0
For CAS Action Set for Cardinality ...
   Custom version information: 1.01
For CAS Action Set for DS2 ...
   Custom version information: 1.0
For CAS Action Set for Decision Tree ...
   Custom version information: 1.01
For Action Set for FEDSQL ...
   Custom version information: 1.0
For CAS Action Set for simple one-off Time Series ...
   Custom version information: 1.0
For CAS Action Set for K-Means Cluster Analytics ...
   Custom version information: 1.01
For CAS Action Set for Streaming Data ...
   Custom version information: 1.0
For CAS Action Set for Data Mining, Machine ...
   Custom version information: 1.0
For CAS Action Set for Artificial Neural Net ...
   Custom version information: 1.01
For CAS Action Set for Nonlinear Models ...
   Custom version information: 1.01
For CAS Action Set for OPTMINER ...
   Custom version information: 1.01
For SAS/Secure 168-bit ...
   Custom version information: V.03.01
For Action Set for Parsing and Categorization ...
   Custom version information: 1.01
```

Chapter 17

PWENCODE Procedure

Overview: PWENCODE Procedure

The PWENCODE procedure enables you to encode passwords. Encoded passwords can be used in place of plaintext passwords in SAS programs that access relational database management systems (RDBMSs) and various servers. An example is a SAS/CONNECT server.

Concepts: PWENCODE Procedure

Using Encoded Passwords in SAS Programs

When a password is encoded with PROC PWENCODE, the output string includes a tag that identifies the string as having been encoded. An example of a tag is **{sas001}**. The tag indicates the encoding method. SAS servers recognize the tag and decode the string before using it. Encoding a password enables you to write SAS programs without having to specify a password in plaintext.

Note: PROC PWENCODE passwords can contain up to a maximum of 512 characters, which include alphanumeric characters, spaces, and special characters. Data set passwords, however, must follow SAS naming rules.

The encoded password is never written to the SAS log in plain text. Instead, each character of the password is replaced by an X in the SAS log.

Why Use PROC PWENCODE

PROC PWENCODE uses encoding to disguise passwords. With encoding, one character set is translated to another character set through some form of table lookup.

PROC PWENCODE is intended to prevent casual, non-malicious viewing of passwords. You should not depend on PROC PWENCODE for all your data security needs; a determined and knowledgeable attacker can decode the encoded passwords.

Syntax: PWENCODE Procedure

Restriction: This procedure is not supported by the CAS engine.

PROC PWENCODE IN='*password*' <OUT=*fileref*> <METHOD=*encoding-method*>;

Statement	Task	Example
PROC PWENCODE	Encode a password	Ex. 1, Ex. 2, Ex. 3

PROC PWENCODE Statement

Encodes a password.

Examples:
"Example 1: Encoding a Password" on page 372
"Example 2: Saving an Encoded Password to the Paste Buffer" on page 372
"Example 3: Specifying Method= SAS003 to Encode a Password" on page 373

Syntax

PROC PWENCODE IN='*password*' <OUT=*fileref*> <METHOD=*encoding-method*>;

Required Argument

IN='*password*'
specifies the password to encode. The password can contain up to a maximum of 512 characters, which include alphanumeric characters, spaces, and special characters.

Note: Data set passwords must follow SAS naming rules. If the IN=*password* follows SAS naming rules, it can also be used for SAS data sets.

If the password contains embedded single or double quotation marks, use the standard SAS rules for quoting character constants.

Note: Each character of the encoded password is replaced by an X when written to the SAS log.

See "Example 1: Encoding a Password" on page 372

"Example 2: Saving an Encoded Password to the Paste Buffer" on page 372

Optional Arguments

OUT=*fileref*

specifies a fileref to which the output string is to be written. If the OUT= option is not specified, the output string is written to the SAS log.

Note: The global macro variable

`_PWENCODE`

is set to the value that is written to the OUT= fileref or to the value that is displayed in the SAS log.

METHOD=*encoding-method*

specifies the encoding method. **SAS002**–**SAS004** encoding methods use fixed keys that cannot be changed. Here are the supported values for *encoding-method*:

Table 17.1 *Supported Encoding Methods*

Encoding Method	Description	Supported Algorithm
sas001	Uses base64 to encode passwords.	None
sas002, which can also be specified as **sasenc**	Uses a 32-bit key to encode passwords.	SASProprietary, which is supported in SAS software.
sas003	Uses a 256-bit key plus 16-bit salt to encode passwords.	AES (Advanced Encryption Standard), which is supported in SAS software.
sas004	Uses a 256-bit key plus 64-bit salt value to encode passwords.	AES (Advanced Encryption Standard), which is supported in SAS software.

Encoding method **sas002** is the default method used. If the METHOD= option is omitted, the default encoding method is used.

The **SAS003** and the **SAS004** encoded passwords consist of a 256-bit key plus a salt value. The key is fixed and cannot be changed.

For **SAS003**, you have a 16-bit salt. For **SAS004**, you have a 64-bit salt. The salt values are random. Therefore, each time you use PROC PWENCODE to encode the same password, you get a different salt value and therefore a different encoded password.

Examples: PWENCODE Procedure

Example 1: Encoding a Password

Features: IN= argument

Details

This example shows a simple case of encoding a password and writing the encoded password to the SAS log.

Program

```
proc pwencode in='my password';
run;
```

Program Description

Encode the password.

```
proc pwencode in='my password';
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log.

```
19   proc pwencode in=XXXXXXXXXXXXX;
20   run;


{SAS002}DBCC571245AD0B31433834F80BD2B99E16B3C969

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Example 2: Saving an Encoded Password to the Paste Buffer

Features: IN= argument
OUT= option

Other features: FILENAME statement with CLIPBRD access method

DETAILS

This example saves an encoded password to the paste buffer. You can then paste the encoded password into another SAS program or into the password field of an authentication dialog box.

Program

```
filename clip clipbrd;

proc pwencode in='my password' out=clip;
run;
```

Program Description

Declare a fileref with the CLIPBRD access method.

```
filename clip clipbrd;
```

Encode the password and save it to the paste buffer. The OUT= option saves the encoded password to the fileref that was declared in the previous statement.

```
proc pwencode in='my password' out=clip;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log.

```
24
25   filename clip clipbrd;
26     proc pwencode in=XXXXXXXXXXXXX out=clip;
27   run;

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Example 3: Specifying Method= SAS003 to Encode a Password

Features: METHOD= argument

Details

This example shows a simple case of encoding a password using the **sas003** encoding method and writing the encoded password to the SAS log.

Program

```
proc pwencode in='my password' method=sas003;
run;
```

Program Description

Encode the password using SAS003.

```
proc pwencode in='my password' method=sas003;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log. **SAS003** uses AES with a 256-bit key plus a 16-bit salt. Because **SAS003** uses random salting, each time you run the following code, a different password is generated.

```
8   proc pwencode in=XXXXXXXXXXXXX method=sas003;
29   run;

{SAS003}08D7B93810D390916F615117D71B2639B4BE

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Chapter 18

SORT Procedure

Overview: SORT Procedure

What Does the SORT Procedure Do?

The SORT procedure orders SAS data set observations by the values of one or more character or numeric variables. The SORT procedure either replaces the original data set or creates a new data set. PROC SORT produces only an output data set. For more information, see "Procedure Output" on page 406.

Note: If extended attributes are defined on the input data set, PROC SORT propagates the extended attributes to the output data set.

The SORT procedure uses the sort utility that is specified by the SORTPGM system option. Sorting can be done by SAS or by the **syncsort** utility. You can use all of the options that are available to the SAS sort utility, such as the SORTSEQ and NODUPKEY options. In some situations, you can improve your performance by using the NOEQUALS option. If you specify an option that is not supported by the host sort, then the SAS sort is used instead.

Sorting SAS Data Sets

In the following example, the original data set was in alphabetical order by last name. PROC SORT replaces the original data set with a data set that is sorted by employee identification number. The following log shows the results from running this PROC SORT step. shows the results of the PROC PRINT step. The statements that produce the output follow:

```
proc sort data=employee;
   by idnumber;
run;

proc print data=employee;
run;
```

Log 18.1 *SAS Log Generated by PROC SORT*

```
NOTE: There were six observations read from the data set WORK.EMPLOYEE.
NOTE: The data set WORK.EMPLOYEE has six observations and three variables.
NOTE: PROCEDURE SORT used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Output 18.1 *Observations Sorted by the Values of One Variable*

```
                    The SAS System                          1

              Obs    Name          IDnumber

               1     Belloit         1988
               2     Wesley          2092
               3     Lemeux          4210
               4     Arnsbarger      5466
               5     Pierce          5779
               6     Capshaw         7338
```

The following output shows the results of a more complicated sort by three variables. The businesses in this example are sorted by town, then by debt from highest amount to lowest amount, then by account number. For an explanation of the program that produces this output, see "Example 2: Sorting in Descending Order" on page 409.

Output 18.2 *Observations Sorted by the Values of Three Variables*

```
                 Customers with Past-Due Accounts                      1
              Listed by Town, Amount, Account Number

                                                          Account
     Obs    Company                   Town              Debt      Number

       1    Paul's Pizza              Apex              83.00      1019
       2    Peter's Auto Parts        Apex              65.79      7288
       3    Watson Tabor Travel       Apex              37.95      3131
       4    Tina's Pet Shop           Apex              37.95      5108
       5    Apex Catering             Apex              37.95      9923
       6    Deluxe Hardware           Garner           467.12      8941
       7    Boyd & Sons Accounting    Garner           312.49      4762
       8    World Wide Electronics    Garner           119.95      1122
       9    Elway Piano and Organ     Garner            65.79      5217
      10    Ice Cream Delight         Holly Springs    299.98      2310
      11    Tim's Burger Stand        Holly Springs    119.95      6335
      12    Strickland Industries     Morrisville      657.22      1675
      13    Pauline's Antiques        Morrisville      302.05      9112
      14    Bob's Beds                Morrisville      119.95      4998
```

Concepts: SORT Procedure

Threaded Sorting

The THREADS system option enables threaded sorting. Threaded sorting achieves a degree of parallelism in the sorting operations. This parallelism is intended to reduce the real time to completion for a given operation and therefore limit the cost of additional CPU resources.

The multi-threaded SAS sort can also be invoked when you specify the THREADS option in the PROC SORT statement. The multi-threaded sort stores all temporary data in a single utility file within one of the locations that are specified by the UTILLOC=

system option. The size of this utility file is proportional to the amount of data that is read from the input data set. A second utility file of the same size can be created in another of these locations when the amount of data that is read from the input data set is large or the amount of memory that is available to the SORT procedure is small. For more information, refer to "UTILLOC= System Option" in *SAS Viya System Options: Reference*.

Note: The TAGSORT option on page 398 does not support threaded sorting.

The multi-threaded SAS sort can be invoked when the THREAD system option is specified and the value of the CPUCOUNT= system option is greater than 1. The value of the SAS system option CPUCOUNT= affects the performance of the threaded sort. CPUCOUNT= suggests how many system CPUs are available for use by the threaded procedures.

For more information, see the "THREADS System Option" in *SAS Viya System Options: Reference* and the "CPUCOUNT= System Option" in *SAS Viya System Options: Reference*.

Sorting Orders for Numeric Variables

For numeric variables, the following is the smallest-to-largest comparison sequence:

1. SAS missing values (shown as a period or special missing value)
2. negative numeric values
3. zero
4. positive numeric values

Sorting Orders for Character Variables

Default Collating Sequence

The order in which alphanumeric characters are sorted is known as the collating sequence. This sort order is determined by the session encoding, which must be UTF-8 for the current release.

By default, PROC SORT uses the ASCII collating sequence when it compares character values.

For more information about the various collating sequences and when they are used, see "Collating Sequence" in *SAS Viya National Language Support: Reference Guide*.

ASCII Order

From the smallest to the largest character that you can display, the English-language ASCII sequence is consistent with the order shown in the following table.

Table 18.1 *ASCII Sort Order Example*

blank ! " # $ % & ' ( ) * + , - . /0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z[\] ◦_
a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~

The main features of the ASCII sequence are that digits are sorted before uppercase letters, and uppercase letters are sorted before lowercase letters. The blank is the smallest character that you can display.

Specifying Sorting Orders for Character Variables

The options ASCII, DANISH, FINNISH, NATIONAL, NORWEGIAN, SWEDISH, and REVERSE specify collating sequences that are stored in the HOST catalog. These can be specified as shown here for DANISH:

```
proc sort data=sample danish;
  by myvar;
run;
```

Additional collating sequences are available, including ITALIAN, POLISH, and SPANISH, but these options require the SORTSEQ= option. Use the SORTSEQ= option as shown here for ITALIAN:

```
proc sort data=sample sortseq=italian;
   by myvar;
run;
```

You can use the SORTSEQ= option for all collating sequences, including those in the HOST catalog. However, some restrictions apply when you use SORTSEQ=. For more information, see "Collating-Sequence Options" on page 387.

If you want to provide your own collating sequences or change a collating sequence provided for you, then use the TRANTAB procedure to create or modify translation tables. When you create your own translation tables, they are stored in your PROFILE catalog, and they override any translation tables that have the same name in the HOST catalog.

Note: System managers can modify the HOST catalog by copying newly created tables from the PROFILE catalog to the HOST catalog. Then all users can access the new or modified translation table.

Linguistic Collation sorts data according to rules of language. For detailed information about Linguistic Collation, see "Collating Sequence" in *SAS Viya National Language Support: Reference Guide*.

Creating Your Own Collating Sequences

If you want to provide your own collating sequences or change a collating sequence provided for you, use the TRANTAB procedure to create or modify translation tables. When you create your own translation tables, they are stored in your Sasuser.Profile catalog, and they override any translation tables by the same name that are stored in the Host catalog.

Note: System managers can modify the Host catalog by copying newly created tables from the Profile catalog to the Host catalog. Then, all users can access the new or modified translation table.

To see the names of the collating sequences that are stored in the Host catalog, issue the following statements:

```
proc catalog catalog=sashelp.host;
contents;
run;
```

Entries of type TRANTAB are the collating sequences.

To see the contents of a particular translation table, use the following statements:

```
proc trantab table=table-name;
list;
run;
```

The contents of collating sequences are displayed in the SAS log.

Stored Sort Information

PROC SORT records the BY variables, collating sequence, and character set that it uses to sort the data set. This information is stored with the data set to help avoid unnecessary sorts.

Before PROC SORT sorts a data set, it checks the stored sort information. If you try to sort a data set how it is currently sorted, then PROC SORT does not perform the sort and writes a message to the log to that effect. To override this behavior, use the FORCE option. If you try to sort a data set the same way it is currently sorted and you specify an OUT= data set, then PROC SORT simply makes a copy of the DATA= data set.

To override the sort information that PROC SORT stores, use the _NULL_ value with the SORTEDBY= data set option. Refer to the "SORTEDBY= Data Set Option" in *SAS Viya Data Set Options: Reference*.

If you want to change the sort information for an existing data set, then use the SORTEDBY= data set option in the MODIFY statement in the DATASETS procedure.

To access the sort information that is stored with a data set, use the CONTENTS statement in PROC DATASETS. For more information, see "CONTENTS Statement" on page 48.

The number of variables by which you can sort a data set with PROC SORT is limited only by available memory. The number of columns by which you can order the rows of a result set using PROC SQL, is also limited only by available memory. The sort indicator, whether stored in the metadata of a SAS data set or represented in memory, is limited to 127 variables. For this reason, up to 127 variables can be stored in the sort indicator or listed on the SORTEDBY= data set option. If you are sorting by more than 127 variables, then only the first 127 are recorded in the sort indicator. If you sort the data set again by the entire list of BY variables, the data set is not recognized as being sorted, because the additional variables (beyond 127) are not found within the sort indicator.

Presorted Input Data Sets

Specifying the "PRESORTED" option prevents SAS from sorting an already sorted data set. Before sorting, SAS checks the sequence of observations within the input data set to determine whether the observations are in order. Use the PRESORTED option when you know or strongly suspect that a data set is already in order according to the key variables specified in the BY statement. The sequence of observations within the data set is checked by reading the data set and comparing the BY variables of each observation read to the BY variables of the preceding observation. This process continues until either the entire data set has been read or an out-of-sequence observation is detected.

If the entire data set has been read and no out-of-sequence observations have been found, then one of two actions is taken. If no output data set has been specified, the sort order metadata of the input data set is updated to indicate that the sequence has been verified. This verification notes that the data set is validly sorted according to the specified BY variables. Otherwise, if the observation sequence has been verified and an output data set is specified, the observations from the input data set are copied to the output data set. The metadata for the output data set indicates that the data is validly sorted according to the BY variables.

If observations within the data set are not in sequence, then the data set is sorted.

If the "NODUPKEY" on page 395 option has been specified, then the sequence checking determines whether observations with duplicate keys are present in the data set. Otherwise, the input data set is deemed not to be sorted if the NODUPKEY option is specified and observations with duplicate keys are detected.

If the metadata of the input data set indicates that the data is already sorted according to the key variables listed in the BY statement and the input data set has been validated, then neither sequence checking nor sorting is performed.

For more information, see "SORTVALIDATE System Option" in *SAS Viya System Options: Reference*.

Linguistic Sorting of Data Sets and ICU

Linguistic collation sorts characters in a culturally sensitive manner according to rules that are associated with a language and locale. The rules and default collating sequence are based on the language specified in the current locale setting. The implementation is provided by the International Components for Unicode (ICU) library. It produces results that are largely compatible with the Unicode Collation Algorithms (UCA).

SAS provides ICU collation when the linguistic option (SORTSEQ=LINGUISTIC) is specified on the SAS procedure, PROC SORT. You can specify linguistic collation using the SORTSEQ= option in the SQL procedure and specify the SORTSEQ=LINGUISTIC system option.

Note: Only PROC SORT and PROC SQL are affected when the SORTSEQ=LINGUISTIC system option is specified.

When the SORTSEQ=LINGUISTIC option is specified, SAS relies on the ICU libraries as the reference implementation of the Unicode Collation Algorithm (UCA) and as a de facto standard. For in-depth information about the UCA algorithm or the International Components for Unicode (ICU) library implementation, see Download the ICU 4.8 Release and CLDR 2.0 Release Note.

The ICU library, incorporated by SAS and used by PROC SORT, is version 4.8.1. This ICU version uses locale data from version 2.0 of the Unicode Common Locale Data Repository (CLDR).

A change in the version of the ICU that is used by PROC SORT for linguistic collation can affect the interpretation of data sets sorted by another version of SAS. If a data set is linguistically sorted by one or more character variables in one version of SAS, the data set is recognized as being sorted when accessed in another version of SAS if the two SAS versions use different versions of the ICU. Because collation rules can change between ICU versions, variations in the rules can cause the order of observations produced by PROC SORT to be different. If the ordering differences are ignored, unexpected results can be seen during processing.

When sorting linguistically, the ICU version used by SAS is recorded in the sort indicator that is stored in the data set header. The ICU version is examined when determining if a data set is considered sorted. A difference between the ICU version in use and the ICU version recorded in the sort indicator of a data set causes the SAS system to ignore the indicated sort order and assume that the data set is unsorted.

Note: The PROC CONTENTS output shows the ICU version in use.

If a sort indicator on a permanent data set is ignored, to facilitate processing, it can be desirable to reassert the order and reestablish the sort indicator on the data set. This can be done using PROC SORT with the PRESORTED option. Most often, because the order of observations within the data set has not been disturbed and is likely correct, the

SORT procedure probably only needs to sequentially read the data set to reestablish the indicator instead of performing a complete sort. If the order of observations is not correct, then the SORT procedure reorders the observations as necessary.

For the COPY procedure, if the ICU version recorded on an input data set is different from the version in use by the SAS system, then the sort indicator on the input data set is ignored, the output data set is not marked as sorted, and a message is written to the SAS log. However, this procedure writes observations to an output data set in the same order as they are read from the input. This order is preserved if a physical order is supported by the engine used for the OUT= destination library. For these reasons, consider re-establishing the sort order of permanent data sets using PROC SORT with the PRESORTED option.

Additional information about how linguistic collation is used by SAS can be found in the following documents, as well as in the PROC SORT SORTSEQ=LINGUISTIC system option.

- See "SORTSEQ=set–table | LINGUISTIC" in *SAS Viya SQL Procedure User's Guide*.
- See PROC SORT option" LINGUISTIC<*(collating-options)*>" on page 389.
- See "Specifying Linguistic Collation" in *SAS Viya National Language Support: Reference Guide*.
- See Chapter 2, "CONTENTS Procedure," on page 13.
- See Chapter 3, "COPY Procedure," on page 33.
- See the Appendix 2, "ICU License," on page 495.

The following are SAS papers that provide detailed information about Linguistic Collation.

- Creating Order out of Character Chaos: Collation Capabilities of the SAS System
- A Sampler of What's New in Base SAS 9.2
- Linguistic Collation: Everyone Can Get What They Expect
- Processing Multilingual Data with the SAS 9.2 Unicode Server
- New Language Features in SAS 9.2 for the Global Enterprise

The following is a list of third-party documentation that should be read for in-depth information about Linguistic Collation.

- See the Unicode Collation Algorithm (UCA) Specification.
- See the Collation section of the ICU User Guide
- For information about the collation rules that are shown in the ICU Locale Explorer, see ICU Locale Explorer. From the Locale Explorer, you can run a demo that enables you to sort a list of words using various collation options. For example, see Collation Rules for English (United States).

Performance Tuning for PROC SORT

Improving Performance with the SORTSIZE= Option

The SORTSIZE system option limits the amount of memory that is available to PROC SORT. In general, you should set the SORTSIZE= option to be no larger than the amount of memory that is available to the SAS process through the MEMSIZE option.

When the SORTSIZE= value is large enough to fit the entire data set in memory, you can achieve optimal sort performance provided that your computer system has the same SORTSIZE= value of physical RAM free. If you do not have enough physical RAM, then your computer starts swapping the extra memory pages to disk and negates the performance gains of using memory.

If the entire data set to be sorted does not fit in the memory space that is allocated by SORTSIZE, SAS creates a temporary utility file to store the data. In this case, SAS uses a sort algorithm that is tuned to sort using disk space instead of memory. These temporary utility files are placed in the SAS WORK location, but these files can be pointed to a different file system so that I/O is not impeded when you use the UTILLOC system option.

If you can place the SAS data file that you want to sort in physical memory on your machine, then a sort in SAS is very efficient. Set SORTSIZE to be larger than the size of the data file. If you cannot fit the data file in physical memory, then set SORTSIZE to 1G or less. In addition, SORTSIZE should always be set to a value that is at least 8M smaller than MEMSIZE.

Note: You can also use the SORTSIZE system option, which has the same effect as the SORTSIZE= option, in the PROC SORT statement.

How SAS Determines the Amount of Memory to Use

The MEMSIZE system option limits the amount of memory that is available to the SAS process. The SORTSIZE system option limits the amount of memory that is available to PROC SORT. The REALMEMSIZE system option specifies the amount of real (not virtual) memory that is made available to SAS.

Although memory settings below the default values for MEMSIZE and SORTSIZE might adversely affect sorting and SAS performance, making large amounts of memory available might be of no benefit. The key for determining whether additional memory might improve performance is whether the sort fits in memory. If the sorted file requires more memory than is allocated, then a SORTSIZE value in the range of 64–512M is generally the optimal value. SORTSIZE should always be set to a value that is at least 8M smaller than MEMSIZE.

For information about setting the REALMEMSIZE system option, see "REALMEMSIZE System Option" in *SAS Viya System Options: Reference*.

Note: If you receive an out of memory error, then increase the value of MEMSIZE. For more information, see "MEMSIZE System Option" in *SAS Viya System Options: Reference*.

Guidelines for Setting the REALMEMSIZE System Option

You can use the REALMEMSIZE system option with PROC SORT to determine how much memory to use. It is important that the REALMEMSIZE value reflects the amount of memory that is available on your system. For optimal performance, the maximum value for the memory setting for all of your applications (including file cache), should never exceed the amount of physical RAM on your computer. The default value for REALMEMSIZE is 80% of the MEMSIZE setting. If REALMEMSIZE is set too high, then PROC SORT might use more memory than is actually available. Using too much memory causes excessive paging and adversely impact system performance.

In general, REALMEMSIZE should be set to the amount of physical memory (not including swap space) that you expect to be available to SAS at run time. A good starting value is the amount of physical memory installed on the computer less the amount that is being used by running applications and the operating system. You can experiment with the REALMEMSIZE value until you reach optimum performance for

your environment. In some cases, optimum performance can be achieved with a very low REALMEMSIZE value. A low value could cause SAS to use less memory and leave more memory for the operating system to perform I/O caching.

For more information, see "REALMEMSIZE System Option" in *SAS Viya System Options: Reference*.

Using Other Options That Affect Performance

The THREADS system option controls whether threaded procedures use threads. It is available as both a system option and as a procedural override in PROC SORT.

The CPUCOUNT option is directly related to the THREADS option and defaults to the number of CPUs on your computer. Depending on your file system and the number of concurrent users, you might benefit from lowering the CPUCOUNT on machines that have many CPUs. When the value of CPUCOUNT equals ACTUAL, SAS returns the number of physical CPUs that are associated with the operating environment where SAS is executing.

The UTILLOC system option allows for the spreading of utility files, and is a good option for balancing I/O.

The DETAILS option, specified in the PROC SORT statement, causes PROC SORT to write messages to the SAS log detailing whether the sort was performed in memory. If the sort was not performed in memory, then the details that are written include the number of utility files and their sizes.

For more information about the THREADS, CPUCOUNT, and UTILLOC system options see *SAS Viya System Options: Reference*.

Disk Space Considerations for PROC SORT

You need to consider the following information when determining the amount of disk space needed to run PROC SORT:

input SAS data set
: PROC SORT uses the SAS input data set specified by the DATA= option.

output SAS data set
: PROC SORT stores the output SAS data set in the location that is specified by the OUT= option. If you use the SAS single-threaded sort, and the OUT= option is not specified, PROC SORT stores the output SAS data set in the Work library.

utility file
: The UTILLOC system option affects the storage location of the utility file only when the SAS multi-threaded sort is used. The SAS single-threaded sort still stores its utility file in the **`Work`** directory. Generally, for the single-threaded sort, the utility file is slightly larger than the uncompressed input SAS data set because additional sortkey data, derived from the BY variables, is included with each record. The utility file can be significantly larger than the uncompressed input SAS data set when BY variables comprise a large portion of an observation or when you use the SORTSEQ=LINGUISTIC option with character BY variables. The utility file can also double in size in extreme circumstances, such as when you have a very large input data set, very little memory available for sorting, or a large utility file page size.

  When the SORT procedure invokes the multi-threaded sort, you can distribute multiple utility files to different locations. The utility file is similar in size to the uncompressed input SAS data set. Usually, only a single utility file of this size is required. However, in extreme circumstances, there might be up to two utility files of

this size that are used. PROC SORT distributes the two utility files to the next two and least recently used locations.

Note: You can use the UTILLOC system option to specify a location in which applications can store utility files.

temporary output SAS data set
During the sort, PROC SORT creates its output in the directory specified in the OUT= option (or directory of the input SAS data set if the OUT= option is not specified). The temporary data set has the same filename as the original data set, except it has an extension of .lck. After the sort completes successfully, the original data set is deleted, and the temporary data set is renamed to match the original data set. Therefore, you need to have enough available disk space in the target directory to hold two copies of the data set.

You can reduce the amount of disk space that is needed by specifying the OVERWRITE option in the PROC SORT statement. When OVERWRITE is specified, SORT, if possible, deletes the input data set before it attempts to write the replacement output data set. Deleting the input data set first can free storage space. This option should be used only with a data set that is backed up, or with a data set that you can reconstruct. For more information, see Chapter 18, "SORT Procedure," on page 376.

Syntax: SORT Procedure

Restriction: The DATA IN= and OUT= options cannot point to CAS simultaneously in SAS Viya. A libname reference to CAS is not supported for both options simultaneously. Here is an example that is not supported:

```
PROC SORT DATE=CAS.FOO OUT=CAS.FOO
```

You can point to CAS with one of the options. For example:

```
PROC SORT DATA=CAS.FOO OUT=WORK.BAR
```

Requirement: BY statement

Tips: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements with the PROC SORT procedure.

For in-database processing to occur, your data must reside within a supported version of a DBMS that has been properly configured for SAS in-database processing. For more information, see "In-Database Processing: PROC SORT" on page 402.

PROC SORT <*collating-sequence-option*> <*other option(s)*>;
BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* …>;
KEY *variable(s)* </ *option*>;

Statement	Task	Example
PROC SORT	Order SAS data set observations by the values of one or more character or numeric variables	Ex. 1, Ex. 4, Ex. 5
BY	Specify the sorting variables	Ex. 1, Ex. 2, Ex. 5
KEY	Specify sorting keys and variables	

PROC SORT Statement

Orders SAS data set observations by the values of one or more character or numeric variables.

Examples: "Example 1: Sorting by the Values of Multiple Variables" on page 407
"Example 4: Maintaining the Relative Order of Observations in Each BY Group" on page 412
"Example 5: Retaining the First Observation of Each BY Group" on page 415

Syntax

PROC SORT <*collating-sequence-option*> <*other option(s)*>;

Summary of Optional Arguments

DATA= *SAS-data-set*
specifies the input data set.

DATECOPY
sorts a SAS data set without changing the created and modified dates.

FORCE
forces redundant sorting.

OVERWRITE
deletes the input data set before the replacement output data set is populated.

PRESORTED
specifies whether the data set is likely already sorted.

SORTSIZE=*memory-specification*
specifies the available memory.

TAGSORT
reduces temporary disk usage.

Create output data sets

DUPOUT= *SAS-data-set*
specifies the output data set to which duplicate observations are written.

OUT= *SAS-data-set*
specifies the output data set.

UNIQUEOUT= *SAS-data-set*
specifies the output data set for eliminated observations.

Eliminate duplicate observations

NODUPKEY
deletes observations with duplicate BY values.

Eliminate unique observations

NOUNIQUEKEY
eliminates observations from the output data set that have a unique sort key.

Override SAS system option THREADS

NOTHREADS
prevents threaded sorting.

THREADS | NOTHREADS
enables or prevents the activation of threaded sorting.

Specify the collating sequence

ASCII
specifies ASCII.

DANISH
specifies Danish.

FINNISH
specifies Finnish.

NATIONAL
specifies a customized sequence.

NORWEGIAN
specifies Norwegian.

REVERSE
reverses the collation order for character variables.

SORTSEQ= *collating-sequence*
specifies the collating sequence.

SWEDISH
specifies Swedish.

Specify the output order

EQUALS | NOEQUALS
specifies the relative order within BY groups.

NOEQUALS
does not maintain relative order within BY groups.

Collating-Sequence Options

Operating Environment Information
For information about behavior specific to your operating environment for the DANISH, FINNISH, NORWEGIAN, or SWEDISH *collating-sequence-option*, see the SAS documentation for your operating environment.

You can specify only one *collating-sequence-option* and multiple *other options* in a PROC SORT step. The order of the two types of options does not matter and both types are not necessary in the same PROC SORT step.

ASCII
sorts character variables using the ASCII collating sequence. You need this option only when you want to achieve an ASCII ordering on a system where EBCDIC is the native collating sequence.

See "ASCII Order" on page 378

DANISH
sorts characters according to the Danish and Norwegian convention.

The Danish and Norwegian collating sequence is shown in Figure 18.1 on page 388.

FINNISH
sorts characters according to the Finnish and Swedish convention.

The Finnish and Swedish collating sequence is shown in Figure 18.1 on page 388.

NATIONAL
sorts character variables using an alternate collating sequence, as defined by your installation, to reflect a country's National Use Differences. To use this option, your site must define a customized national sort sequence. Check with the SAS Installation Representative at your site to determine whether a customized national sort sequence is available.

NORWEGIAN
sorts characters according to the Danish and Norwegian convention.

The Norwegian collating sequence is shown in Figure 18.1 on page 388.

REVERSE
sorts character variables using a collating sequence that is reversed from the normal collating sequence.

Restriction Only one collating-sequence-option can be specified.

Interaction Using REVERSE with the DESCENDING option in the BY statement restores the sequence to the normal order.

See The "DESCENDING" on page 400 option in the BY statement. The difference is that the DESCENDING option can be used with both character and numeric variables.

SWEDISH
sorts characters according to the Finnish and Swedish convention.

The Finnish and Swedish collating sequence is shown in Figure 18.1 on page 388.

SORTSEQ= *collating-sequence*
The *collating-sequence* can be one of the following:

- collating-sequence-option on page 388
- translation_table on page 388
- encoding-value on page 389
- LINGUISTIC on page 389

For detailed information, refer to "Collating Sequence" in *SAS Viya National Language Support: Reference Guide*.

Figure 18.1 *National Collating Sequences of Alphanumeric Characters*

```
Danish:     0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Finnish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
Italian:    0123456789AÀBCÇDEÉÈFGHIÌJKLMNOÒPQRSTUÙVWXYZaàbcçdeéèfghiìjklmnoòpqrstuùvwxyz
Norwegian:  0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Spanish:    0123456789AÁaáBbCcDdEÉeéFfGgHhIÍiíJjKkLlMmNnÑñOÓoóPpQqRrSsTtUÚuúÜüVvWwXxYyZz
Swedish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
```

Here are descriptions of the types of collating sequences:

collating-sequence-option | translation_table
specifies one of the PROC SORT statement *collating-sequence-options* (ASCII, DANISH, FINNISH, NATIONAL, NORWEGIAN, REVERSE, SWEDISH) or a translation table, which can be one that SAS provides or any user-defined translation table. Translation tables provided by SAS are: ASCII, DANISH,

FINNISH, ITALIAN, NORWEGIAN, POLISH, REVERSE, SPANISH, and SWEDISH.

Restriction You can specify only one *collating-sequence-option* or one translation table for the SORTSEQ= option.

Interaction In-database processing will not occur when the SORTSEQ= option is specified.

Tip The SORTSEQ= collating-sequence options are specified without parenthesis and there are no arguments associated with them.

See For a more detailed description of each *collating-sequence-option*, see "Collating-Sequence Options" on page 387.

To see the Sorting Order of Character variables, "ASCII Order" on page 378 and Figure 18.1 on page 388 for all others.

Example

```
proc sort data=mydata SORTSEQ=ASCII;
```

encoding-value

specifies an encoding value. The result is the same as a binary collation of the character data represented in the specified encoding. See the supported encoding value in the *SAS Viya National Language Support: Reference Guide*.

Restriction PROC SORT is the only procedure or part of the SAS system that recognizes an encoding specified for the SORTSEQ= option.

Tip When the encoding value contains a character other than an alphanumeric character or underscore, the value needs to be enclosed in quotation marks.

See The list of the encodings that can be specified in the *SAS Viya National Language Support: Reference Guide*.

LINGUISTIC<*(collating-options)*>

specifies linguistic collation, which sorts characters in a culturally sensitive manner according to rules that are associated with a language and locale. The rules and default *collating-sequence* options are based on the language that is specified in the current locale setting. The implementation is provided by the International Components for Unicode (ICU) library. It produces results that are largely compatible with the Unicode Collation Algorithms (UCA). For more information, see " Linguistic Sorting of Data Sets and ICU" on page 381.

Note: Only PROC SORT and PROC SQL are affected when the linguistic collation system option is specified.

The following are options that can be used when specifying SORTSEQ=LINGUISTIC. These options modify the linguistic collating sequence:

ALTERNATE_HANDLING=SHIFTED

controls the handling of variable characters like spaces, punctuation, and symbols. When this option is not specified (using the default value Non-Ignorable), differences among these variable characters are of the same importance as differences among letters. If the ALTERNATE_HANDLING option is specified, these variable characters are of minor importance.

Default NON_IGNORABLE

Tip The SHIFTED value is often used in combination with STRENGTH= set to Quaternary. In such a case, spaces, punctuation, and symbols are considered when comparing strings, but only if all other aspects of the strings (base letters, accents, and case) are identical.

See "Example 6: Linguistic Sorting Using ALTERNATE_HANDLING=" on page 417 and "Example 7: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=" on page 419.

CASE_FIRST=
specifies the order of uppercase and lowercase letters. This argument is valid for only TERTIARY, QUATERNARY, or IDENTICAL levels. The following table provides the values and information for the CASE_FIRST argument:

Table 18.2 *Arguments for CASE_FIRST=*

Value	Description
UPPER	Sorts uppercase letters first, then the lowercase letters.
LOWER	Sorts lowercase letters first, then the uppercase letters.

COLLATION=
specifies character ordering. The following table lists the available COLLATION= values.

Note: If you do not select a collation value, then the user's locale-default collation is selected.

Table 18.3 *Values for COLLATION=*

Value	Description
BIG5HAN	Specifies Pinyin ordering for Latin and specifies bug5 charset ordering for Chinese, Japanese, and Korean characters.
DIRECT	Specifies a Hindi variant.
GB21312HAN	Specifies Pinyin ordering for Latin and specifies gb2312han charset ordering for Chinese, Japanese, and Korean characters.
PHONEBOOK	Specifies a telephone-book style for ordering of characters. Select PHONEBOOK only with the German language.
PINYIN	Specifies an ordering for Chinese, Japanese, and Korean characters based on character-by-character transliteration into Pinyin. This ordering is typically used with simplified Chinese.
POSIX	Portable Operating System Interface. This option specifies a "C" locale ordering of characters.

Value	Description
STROKE	Specifies a nonalphabetic writing style ordering of characters. Select STROKE with Chinese, Japanese, Korean, or Vietnamese languages. This ordering is typically used with Traditional Chinese.
TRADITIONAL	Specifies a traditional style for ordering of characters. For example, select TRADITIONAL with the Spanish Language.

LOCALE= ***locale_name***
specifies the locale name in the form of a POSIX name (for example, ja_JP). For a list of locale and POSIX values supported by PROC SORT, see "LOCALE= Values for PAPERSIZE and DFLANG Options" in *SAS Viya National Language Support: Reference Guide*.

Restriction The following Locales are not supported by PROC SORT:
- Afrikaans_SouthAfrica, af_ZA
- Cornish_UnitedKingdom, kw_GB
- ManxGaelic_UnitedKingdom, gv_GB

NUMERIC_COLLATION=
orders integer values within the text by the numeric value instead of characters used to represent the numbers.

Table 18.4 *Values for NUMERIC_COLLATION*

Value	Description
ON	Order numbers by the numeric value. For example, "8 Main St." would sort before "45 Main St.".
OFF	Order numbers by the character value. For example, "45 Main St." would sort before "8 Main St.".

Default OFF

STRENGTH=
The value of strength is related to the collation level. There are five collation-level values. The following table provides information about the five levels. The default value for strength is related to the locale.

Table 18.5 *Values for STRENGTH=*

Value	Type of Collation	Description
PRIMARY or 1	PRIMARY specifies differences between base characters (for example, "a" < "b").	It is the strongest difference. For example, dictionaries are divided into different sections by base character.

Value	Type of Collation	Description
SECONDARY or 2	Accents in the characters are considered secondary differences (for example, "as" < "às" < "at").	A secondary difference is ignored when there is a primary difference anywhere in the strings. Other differences between letters can also be considered secondary differences, depending on the language.
TERTIARY or 3	Upper and lowercase differences in characters are distinguished at the tertiary level (for example, "ao" < "Ao" < "aò"). For an example, see "Example 6: Linguistic Sorting Using ALTERNATE_HANDLING=" on page 417.	A tertiary difference is ignored when there is a primary or secondary difference anywhere in the strings. Another example is the difference between large and small Kana.
QUATERNARY or 4	When punctuation is ignored at level 1-3, an additional level can be used to distinguish words with and without punctuation (for example, "a-b" < "ab" < "aB"). For an example, see "Example 7: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=" on page 419.	The quaternary level should be used if ignoring punctuation is required or when processing Japanese text. This difference is ignored when there is a primary, secondary, or tertiary difference.
IDENTICAL or 5	When all other levels are equal, the identical level is used as a tiebreaker. The Unicode code point values of the Normalization Form D (NFD) form of each string are compared at this level, just in case there is no difference at levels 1-4.	This level should be used sparingly, because code-point value differences between two strings rarely occur. For example, only Hebrew cantillation marks are distinguished at this level.

Alias LEVEL=

Alias UCA

Restriction The SORTSEQ=LINGUISTIC option is available only on the PROC SORT SORTSEQ= option and is not available for the SAS System SORTSEQ= option.

Interaction The ICU version can change in a new SAS release. The order of observations produced when sorting a data set linguistically, using one release of SAS, can be different from the order produced by another release if the two releases use different versions of the ICU. When migrating to a new release of SAS, consider re-establishing the sort order of permanent data sets using PROC SORT with the PRESORTED option. For more details, see " Linguistic Sorting of Data Sets and ICU" on page 381.

Tips The CONTENTS procedure or CONTENTS statement output shows the ICU version number of a data set that is linguistically sorted.

The *collating-options* must be enclosed in parentheses. More than one collating option can be specified.

When BY processing is performed on data sets that are sorted with linguistic collation, the NOBYSORTED system option might need to be specified in order for the data set to be treated properly. BY processing is performed differently than collating sequence processing.

See For ICU License Agreement, see Appendix 2, "ICU License," on page 495.

For more information, see "Specifying Linguistic Collation" in *SAS Viya National Language Support: Reference Guide*. For more information, see " Linguistic Sorting of Data Sets and ICU" on page 381.

CAUTION:

If you use a host sort utility to sort your data, then specifying a translation-table-based collating sequence with the SORTSEQ= option might corrupt the character BY variables.

Interaction In-database processing does not occur when the SORTSEQ= option is specified.

Tip The SORTSEQ= *collating-sequence* options are specified without parenthesis and no arguments are associated with them. Here is an example of how to specify a collating sequence: `proc sort data=mydata SORTSEQ=ASCII;`

Other Options

Options can include one *collating-sequence-option* and multiple other options. The order of the two types of options does not matter and both types are not necessary in the same PROC SORT step.

DATA= *SAS-data-set*
identifies the input SAS data set.

Restriction For in-database processing to occur, the data set must refer to a table residing on the DBMS.

Note PROC SORT supports extended attributes by copying the attributes from the input data set to the output data set.

See

SAS Viya Data Set Options: Reference

DATECOPY
copies the SAS internal date and time at which the SAS data set was created and the date and time at which it was last modified before the sort to the resulting sorted data set. Note that the operating environment date and time are not preserved.

Restriction DATECOPY can be used only when the resulting data set uses the V8 or V9 engine.

DUPOUT= *SAS-data-set*
specifies the output data set to which duplicate observations are written.

Interactions In-database processing does not occur when the DUPOUT= option is specified.

The DUPOUT= and UNIQUEOUT= options are not compatible and cannot be specified simultaneously.

Tips The DUPOUT= option can be used only with the NODUPKEY option. It cannot be combined with the NOUNIQUEKEY option.

If the DUPOUT= data set name that is specified is the same as the INPUT data set name, SAS does not sort or overwrite the INPUT data set. Instead, SAS generates an error message. The FORCE option must be specified in order to overwrite the INPUT data set with the DUPOUT= data set of the same name.

See *SAS Viya Data Set Options: Reference*

EQUALS | NOEQUALS
specifies the order of the observations in the output data set. For observations with identical BY-variable values, EQUALS maintains the relative order of the observations within the input data set in the output data set. NOEQUALS does not necessarily preserve this order in the output data set.

Default EQUALS

Interactions When you use NODUPKEY to remove observations in the output data set, the choice of EQUALS or NOEQUALS can affect which observations are removed.

The EQUALS | NOEQUALS procedure option overrides the default sort stability behavior that is established with the SORTEQUALS | NOSORTEQUALS system option.

The EQUALS option is supported by the threaded sort. However, I/O performance might be reduced when using the EQUALS option with the threaded sort because partitioned data sets are processed as if they consist of a single partition.

The NOEQUALS option is supported by the threaded sort. The order of observations within BY groups that are returned by the threaded sort might not be consistent between runs.

Tip Using NOEQUALS can save CPU time and memory.

FORCE

sorts and replaces an indexed data set when the OUT= option is not specified. Without the FORCE option, PROC SORT does not sort and replace an indexed data set because sorting destroys user-created indexes for the data set. When you specify FORCE, PROC SORT sorts and replaces the data set and destroys all user-created indexes for the data set. Indexes that were created or required by integrity constraints are preserved.

Restriction If you use PROC SORT with the FORCE option on data sets that were created with the Version 5 compatibility engine or with a sequential engine such as a tape format engine, you must also specify the OUT= option.

Tip PROC SORT checks for the sort indicator before it sorts a data set so that data is not sorted again unnecessarily. By default, PROC SORT does not sort a data set if the sort information matches the requested sort. You can use FORCE to override this behavior. You might need to use FORCE if SAS cannot verify the sort specification in the data set option SORTEDBY=. For more information about SORTEDBY= , see the chapter on SAS data set options in *SAS Viya Data Set Options: Reference*.

NODUPKEY

checks for and eliminates observations with duplicate BY values. If you specify this option, then PROC SORT compares all BY values for each observation to the ones for the previous observation that is written to the output data set. If an exact match is found, then the observation is not written to the output data set.

Interactions When you are removing observations with duplicate BY values with NODUPKEY, the choice of EQUALS or NOEQUALS can have an effect on which observations are removed.

In-database sorting occurs when the NODUPKEY option is specified and the system option SQLGENERATION= is assigned a DBMS and the system option SORTPGM=BEST.

Options NODUPKEY and NOUNIQUEKEY are not compatible. If these options are specified together, an error is printed to the SAS log.

Tips Use the EQUALS option with the NODUPKEY option for consistent results in your output data sets.

The DUPOUT= option can be used with the NODUPKEY option. It cannot be combined with the NOUNIQUEKEY option.

Example "Example 5: Retaining the First Observation of Each BY Group" on page 415

NOEQUALS
See "EQUALS | NOEQUALS" on page 394.

NOTHREADS
See "THREADS | NOTHREADS" on page 399.

NOUNIQUEKEY
checks for and eliminates observations from the output data set that have a unique sort key. A sort key is unique when the observation containing the key is the only observation within a BY group.

Note: Unlike NODUPKEY, which writes one observation of a BY group to the output data set and discards all other observations from the BY group, the NOUNIQUEKEY maintains BY group integrity. Either all observations of a BY group are written to the output data set when the BY group consists of two or more observations, or all observations of the BY group are discarded when the BY group consists of a single observation.

Alias NOUNIKEY | NOUNIKEYS | NOUNIQUEKEYS

Interaction Options NODUPKEY and NOUNIQUEKEY are not compatible. If NODUPKEY and NOUNIQUEKEY are specified together, an error is printed to the SAS log.

Tip The UNIQUEOUT= option can be used with the NOUNIQUEKEY option. It cannot be combined with the NODUPKEY option.

See UNIQUEOUT= to direct the observations that have been eliminated to an output data set.

OUT= *SAS-data-set*
names the output data set. If *SAS-data-set* does not exist, then PROC SORT creates it.

CAUTION:
Use care when you use PROC SORT without OUT=. Without the OUT= option, PROC SORT replaces the original data set with the sorted observations when the procedure executes without errors.

Default Without OUT=, PROC SORT overwrites the original data set.

Tips With in-database sorts, the output data set cannot refer to the input table on the DBMS.

You can use data set options with OUT=.

See *SAS Viya Data Set Options: Reference*

Example "Example 1: Sorting by the Values of Multiple Variables" on page 407

OVERWRITE
enables the input data set to be deleted before the replacement output data set of the same name is populated with observations.

CAUTION:
Use the OVERWRITE option only with a data set that is backed up or with a data set that you can reconstruct. Because the input data set is deleted, data is lost if a failure occurs while the output data set is being written.

Restrictions If the OVERWRITE and OUT= options are specified and the OUT= data set name is not the same as the INPUT data set name, SAS does not overwrite the INPUT data set.

The OVERWRITE option has no effect if you also specify the TAGSORT option. You cannot overwrite the input data set because TAGSORT must reread the input data set while populating the output data set.

The OVERWRITE option is supported by the SAS sort and SAS threaded sort only. The option has no effect if you are using a host sort.

Tip Using the OVERWRITE option can reduce disk space requirements.

PRESORTED
before sorting, checks within the input data set to determine whether the sequence of observations is in order. Use the PRESORTED option when you know or strongly suspect that a data set is already in order according to the key variables that are specified in the BY statement. By specifying this option, you avoid the cost of sorting the data set.

Interaction Sequence checking is not performed when the "FORCE" on page 395 option is specified.

Tips You can use the DATA step to import data, from external text files, in a sequence compatible with SAS processing and according to the sort order specified by the combination of SORT options and key variables listed in the BY statement. You can then specify the PRESORTED option if you know or highly suspect that the data is sorted accordingly.

Using the PRESORTED option with ACCESS engines and DBMS data is not recommended. These external databases are not guaranteed to return observations in sorted order unless an ORDER BY clause is specified in a query. Generally, physical ordering is not a concept that external databases use. Therefore, these databases are not guaranteed to return observations in the same order when executing a query multiple times. Physical order can be important for producing consistent, repeatable results when processing data. Without a repeatable data retrieval order, PROC SORT does not guarantee the return of observations in the same order from one PROC SORT execution to another, even when the "EQUALS | NOEQUALS" on page 394 option is used to request sort stability. Without a repeatable retrieval order, the detection and elimination of adjacent duplicate records by PROC SORT can also vary from one PROC SORT execution to another.

See System option "SORTVALIDATE System Option" in *SAS Viya System Options: Reference*.

SORTSIZE=*memory-specification*
specifies the maximum amount of memory that is available to PROC SORT. Valid values for *memory-specification* are as follows:

MAX
specifies that all available memory can be used.

n
specifies the amount of memory in bytes, where *n* is a real number.

***n*K**
specifies the amount of memory in kilobytes, where *n* is a real number.

***n*M**
specifies the amount of memory in megabytes, where *n* is a real number.

***n*G**
specifies the amount of memory in gigabytes, where *n* is a real number.

Specifying the SORTSIZE= option in the PROC SORT statement temporarily overrides the SAS system option. For more information, see "SORTSIZE= System Option" in *SAS Viya System Options: Reference*.

Alias SIZE=

Default the value of the SAS system option SORTSIZE=

Tips Setting the SORTSIZE= option in the PROC SORT statement to MAX or 0, or not setting the SORTSIZE= option, limits the PROC SORT to the available physical memory based on the settings of the SAS system options REALMEMSIZE and MEMSIZE.

For information about the SAS system options REALMEMSIZE and MEMSIZE, see the SAS documentation for your operating environment.

TAGSORT
stores only the BY variables and the observation numbers in temporary files. The TAGSORT option in the PROC SORT statement is useful when there is limited disk space available to sort a large SAS data set. The BY variables and the observation numbers are called *tags*. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order.

Note: The utility file created is much smaller than it would be if the TAGSORT option were not specified.

You must have enough disk space to hold an additional copy of the data set (the output data set) and the utility file that contains the tags. By default, this utility file is stored in the Work library. If this directory is too small, you can change this directory by using the WORK system option. For more information, see "WORK System Option" in *SAS Viya System Options: Reference*.

Note that while using the TAGSORT option might reduce temporary disk use, the processing time could be higher. However, on systems with limited available disk space, the TAGSORT option might enable data sets to be sorted in situations where that would otherwise not be possible.

Restriction The TAGSORT option is not compatible with the OVERWRITE option.

Interaction The TAGSORT option is not supported by the threaded sort.

Tip When the total length of BY variables is small compared with the record length, TAGSORT reduces temporary disk usage considerably. However, processing time might be much higher.

THREADS | NOTHREADS

enables or prevents the activation of threaded sorting.

Default The value of the THREADS | NOTHREADS SAS system option. Note that the default can be overridden using the SORT procedure THREADS | NOTHREADS option.

Restrictions Your site administrator can create a restricted options table. A restricted options table specifies SAS system option values that are established at start-up and cannot be overridden. If the THREADS | NOTHREADS system option is listed in the restricted options table, any attempt to set these system options is ignored and a warning message is written to the SAS log.

If a failure occurs when adding the THREADS | NOTHREADS procedure option using the SPD engine, PROC SORT stops processing and writes a message to the SAS log.

Interactions The PROC SORT THREADS | NOTHREADS options override the SAS system THREADS | NOTHREADS options unless the system option is restricted. (See Restriction.) For more information, see "THREADS System Option" in *SAS Viya System Options: Reference*.

The THREADS system option is honored if PROC SORT determines that threaded processing is deemed to be beneficial. If the value of the SAS system option CPUCOUNT=1, then threaded processing is not beneficial. However, you can specify the PROC SORT THREADS option to force threaded processing when the system option is set to NOTHREADS or when the system option is THREADS and the procedure option is NOTHREADS. This option combination prevents threaded processing and overrides the actions taken that are based on the system options. Note that when threaded sorting is in effect and NOEQUALS is specified, observations within BY groups might be returned in an unpredictable order.

If threaded SAS sort is being used, the UTILLOC= system option affects the placement of utility files. Thread-enabled SAS applications are able to create temporary files that can be accessed in parallel by separate threads. For more information, see "UTILLOC= System Option" in *SAS Viya System Options: Reference*.

The page size of the utility file used by PROC SORT is influenced by the new STRIPESIZE= system option. For more information, see "STRIPESIZE= System Option" in *SAS Viya System Options: Reference*.

The TAGSORT option is not supported by the threaded sort. Specifying the TAGSORT option prevents threaded processing.

See "Threaded Sorting" on page 377 .

UNIQUEOUT= ***SAS-data-set***

specifies the output data set for observations eliminated by the NOUNIQUEKEY option.

Alias UNIOUT=

Interaction The DUPOUT= and UNIOUT= options are not compatible and cannot be specified simultaneously.

Tip The UNIQUEOUT= option can be used with the NOUNIQUEKEY option. It cannot be combined with the NODUPKEY option.

See "NOUNIQUEKEY" on page 396

SAS Viya Data Set Options: Reference

BY Statement

Specifies the sorting variables.

Examples: "Example 1: Sorting by the Values of Multiple Variables" on page 407
"Example 2: Sorting in Descending Order" on page 409
"Example 5: Retaining the First Observation of Each BY Group" on page 415

Syntax

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* …>;

Required Argument

variable
specifies the variable by which PROC SORT sorts the observations. PROC SORT first arranges the data set by the values in ascending order, by default, of the first BY variable. PROC SORT then arranges any observations that have the same value of the first BY variable by the values of the second BY variable in ascending order. This sorting continues for every specified BY variable.

Optional Argument

DESCENDING
reverses the sort order for the variable that immediately follows in the statement so that observations are sorted from the largest value to the smallest value. The DESCENDING keyword modifies the variable that follows it.

Tips In a PROC SORT BY statement, the DESCENDING keyword modifies the variable that follows it.

The THREADS SAS system option is the default as long as the PROC SORT THREADS | NOTHREADS option is unspecified.

Example "Example 2: Sorting in Descending Order" on page 409

KEY Statement

Specifies sorting keys and variables. The KEY statement is an alternative to the BY statement. The KEY statement syntax allows for the future possibility of specifying different collation options for each KEY variable. Currently, the only options allowed are ASCENDING and DESCENDING.

Restriction: The BY statement cannot be used with the KEY statement.

Tip: Multiple KEY statements can be specified.

Syntax

KEY *variable(s)* </ *option*> ;

Required Argument

variable(s)

specifies the variable by which PROC SORT orders the observations. Multiple variables can be specified. Each of these variables must be separated by a space. A range of variables can also be specified. For example, the following code shows how to specify multiple variables and a range of variables:

```
data sortKeys;
   input x1 x2 x3 x4 ;
cards;
   7 8 9 8
   0 0 0 0
   1 2 3 4 ;
run;
proc sort data=sortKeys out=sortedOutput;
   key x1 x2-x4;
run;
```

Multiple KEY statements can also be specified. The first sort key encountered from among all sort keys is considered the primary sort key. Sorting continues for every specified KEY statement and its variables. For example, the following code shows how to specify multiple KEY statements:

```
proc sort data=sortKeys out=sortedOutput;
   key x2;
   key x3;
run;
```

The following code example uses the BY statement to accomplish the same type of sort as the previous example:

```
proc sort data=sortKeys out=sortedOutput;
   by x2 x3;
run;
```

Optional Arguments

ASCENDING

sorts in ascending order the variable or variables that it follows. Observations are sorted from the smallest value to the largest value. The ASCENDING keyword modifies all the variables that precede it in the KEY statement.

Alias	ASC
Default	ASCENDING is the default sort order.

Tip In a PROC SORT KEY statement, the ASCENDING option modifies all the variables that it follows. The option must follow the `/`. In the following example, the x1 variable in the input data set is sorted in ascending order.

```
proc sort data=sortVar out=sortedOutput;
   key x1 / ascending;
run;
```

DESCENDING
reverses the sort order for the variable that it follows in the statement so that observations are sorted from the largest value to the smallest value. The DESCENDING keyword modifies all the variables that it precedes in the KEY statement.

Alias DESC

Default ASCENDING (ASC) is the default sort order.

Tip In a PROC SORT KEY statement, the DESCENDING option modifies the variables that follows it. The option must follow the `/`. In the following example, the x1 and x2 variables in the input data set is sorted in descending order:

```
proc sort data=sortVar out=sortedOutput;
   key x1 x2 / descending;
run;
```

The following example uses the BY statement to accomplish the same type of sort as the previous example:

```
proc sort data=sortVar out=sortedOutput;
   by descending x1 descending x2 ;
run;
```

In-Database Processing: PROC SORT

In-database processing has several advantages over processing within SAS. These advantages include increased security, reduced network traffic, and the potential for faster processing. Increased security is possible because sensitive data does not have to be extracted from the DBMS. Faster processing is possible because data is manipulated locally, on the DBMS, using high-speed secondary storage devices instead of being transported across a relatively slow network connection, because the DBMS might have more processing resources at its disposal, and because the DBMS might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

When the DATA= input data set is stored as a table or view in a database management system (DBMS), the PROC SORT procedure can use in-database processing to sort the data. In-database processing can provide the advantages of faster processing and reduced data transfer between the database and SAS software.

In-database processing for PROC SORT now supports the following database management systems:

- Aster

- DB2
- Greenplum
- HAWQ
- Impala
- Netezza
- Oracle
- SAP HANA
- Teradata

PROC SORT performs in-database processing using SQL explicit pass-through. The pass-through facility uses SAS/ACCESS to connect to a DBMS and to send statements directly to the DBMS for execution. This facility lets you use the SQL syntax of your DBMS. For details, see "Pass-Through Facility for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.

In-database processing is used by PROC SORT when a combination of procedure and system options are properly set. When system option SORTPGM=BEST, system option SQLGENERATION= is set to cause in-database processing, and when the PROC SORT NODUPKEY option is specified, PROC SORT generates a DBMS SQL query that sorts the data. The sorted results can either remain as a new table within the DBMS or can be returned to SAS. To view the SQL queries generated, set the SASTRACE= option.

The SAS system option SORTPGM= can also be used without setting the SQLGENERATION option to instruct PROC SORT to use either the DBMS, SAS, or the HOST to perform the sort. If SORTPGM=BEST is specified, then either the DBMS, SAS, or HOST performs the sort. The observation ordering that is produced by PROC SORT depends on whether the DBMS or SAS performs the sorting.

If the DBMS performs the sort, then the configuration and characteristics of the DBMS sorting program affects the resulting data order. The DBMS configuration settings and characteristics that can affect data order include character collation, ordering of NULL values, and sort stability. Most database management systems do not guarantee sort stability, and the sort might be performed by the DBMS regardless of the state of the SORTEQUALS/NOSORTEQUALS system option and EQUALS/NOEQUALS procedure option.

If you set the SAS system option SORTPGM= to SAS, then unordered data is delivered from the DBMS to SAS and SAS performs the sorting. However, consistency in the delivery order of observations from a DBMS is not guaranteed. Therefore, even though SAS can perform a stable sort on the DBMS data, SAS cannot guarantee that the ordering of observations within output BY groups is the same from one PROC SORT execution to the next. To achieve consistency in the ordering of observations within BY groups, first populate a SAS data set with the DBMS data, and then use the EQUALS or SORTEQUALS option to perform a stable sort.

In-database processing is affected by the following circumstances:

- When PROC SORT options, SORTSEQ=, or DUPOUT=, are specified, no in-database processing occurs.
- For in-database processing, the OUT= procedure option must be specified and the output data set cannot refer to the input table on the DBMS.
- LIBNAME options and data set options can also affect whether in-database processing occurs and what type of query is generated. See "In-Database Procedures" in *SAS/ACCESS for Relational Databases: Reference* for a complete list of these

options. The user can also set OPTIONS MSGLEVEL=I in SAS to see which options prevent or affect in-database processing.

Integrity Constraints: SORT Procedure

Sorting the input data set and replacing it with the sorted data set preserves both referential and general integrity constraints, as well as any indexes that they might require. A sort that creates a new data set does not preserve any integrity constraints or indexes. For more information about implicit replacement, explicit replacement, and no replacement with and without the OUT= option, see "Output Data Set" on page 406.

Specifying the Host Sort Utility

Introduction to Using the Host Sort Utility

SAS supports one host sort utility on Linux called **syncsort**. You can use this sorting application as an alternative sorting algorithm to the SAS sort. SAS determines which sort to use by the values that are set for the SORTNAME, SORTPGM, SORTCUT, and SORTCUTP system options.

Setting the Host Sort Utility as the Sort Algorithm

To specify a host sort utility as the sort algorithm, complete the following steps:

1. Specify the name of the host utility (**syncsort**) in the SORTNAME system option.
2. Set the SORTPGM system option to tell SAS when to use the host sort utility.
   - If you specify SORTPGM=HOST, then SAS uses the host sort utility.
   - If you specify SORTPGM=BEST, then SAS chooses the best sorting method (either the SAS sort or the host sort) for the situation.

Sorting Based on Size or Observations

The sort routine that SAS uses can be based on either the number of observations in a data set, or on the size of the data set. When the SORTPGM system option is set to BEST, SAS uses the first available and pertinent sorting algorithm based on the following order of precedence:

- host sort utility
- SAS sort utility

The SORTCUT system option is based on the number of observations in a data set. The SORTCUTP system option is based on the size of the data set. SAS looks at the values for the SORTCUT and SORTCUTP system options to determine which sort routine to use. If the number of observations is greater than or equal to the value of SORTCUT, SAS uses the host sort utility. If the number of bytes in a data set is greater than the value of SORTCUTP, SAS uses the host sort utility.

If SORTCUT and SORTCUTP are set to zero, SAS uses the SAS sort utility. If you specify both system options, and either condition is met, SAS uses the host sort utility.

When the following OPTIONS statement is in effect, the host sort utility (**syncsort**) is used when the number of observations is 500 or greater:

```
options sortpgm=best sortcut=500;
```

In this example, the host sort utility is used when the size of the data set is greater than 40M:

```
options sortpgm=best sortcutp=40M;
```

For more information about these sort options, see "SORTCUT System Option" in *SAS Viya System Options: Reference*, "SORTCUTP System Option" in *SAS Viya System Options: Reference*, and "SORTPGM System Option" in *SAS Viya System Options: Reference*.

Changing the Location of Temporary Files Used by the Host Sort Utility

By default, the host sort utilities use the location that is specified in the -WORK option for temporary files. To change the location of these temporary files, specify a location by using the SORTDEV system option. Here is an example:

```
options sortdev="/tmp/host";
```

For more information, see "SORTDEV System Option" in *SAS Viya System Options: Reference*.

Passing Options to the Host Sort Utility

To specify options for the sort utility, use the SORTANOM system option. For a list of valid options, see "SORTANOM System Option" in *SAS Viya System Options: Reference*.

Passing Parameters to the Host Sort Utility

To pass parameters to the sort utility, use the SORTPARM system option. The parameters that you can specify depend on the host sort utility. For more information, see "SORTPARM System Option" in *SAS Viya System Options: Reference*.

Specifying the SORTSEQ= Option with a Host Sort Utility

The SORTSEQ= option enables you to specify the collating sequence for your sort. For a list of valid values, see Chapter 18, "SORT Procedure," on page 376.

CAUTION:

If you are using a host sort utility to sort your data, then specifying the SORTSEQ= option might corrupt the character BY variables if the sort sequence translation table and its inverse are not one-to-one mappings. In other words, for the sort to work, the translation table must map each character to a unique weight, and the inverse table must map each weight to a unique character.

If your translation tables do not map one-to-one, then you can use one of the following methods to perform your sort:

- Create a translation table that maps one-to-one. Once you create a translation table that maps one-to-one, you can easily create a corresponding inverse table using the TRANTAB procedure. If your translation table is not mapped one-to-one, then the following note appears in the SAS log when you try to create an inverse table:

  ```
  NOTE:  This table cannot be mapped one to one.
  ```

- Use the SAS sort. You can specify the SAS sort using the SORTPGM system option. For more information, see "SORTPGM System Option" in *SAS Viya System Options: Reference*.
- Specify the collation order options of your host sort utility. See the documentation for your host sort utility for more information.
- Create a view with a single BY variable. For an example, see "Example 3: Creating a View with a Single BY Variable" on page 411.

Note: After using one of these methods, you might need to perform subsequent BY processing using either the NOTSORTED option or the NOBYSORTED system option. For more information about the NOTSORTED option, see "BY Statement" in *SAS Viya Statements: Reference*. For more information about the NOBYSORTED system option, see "BYSORTED System Option" in *SAS Viya System Options: Reference*.

Results: SORT Procedure

Procedure Output

PROC SORT produces only an output data set. To see the output data set, you can use PROC PRINT, PROC REPORT, or another of the many available methods of printing in SAS.

Output Data Set

Without the OUT= option, PROC SORT replaces the original data set with the sorted observations when the procedure executes without errors. When you specify the OUT= option using a new data set name, PROC SORT creates a new data set that contains the sorted observations.

Table 18.6 *Data Set Replacement Options*

Task	Options
implicit replacement of input data set	proc sort data=names;
explicit replacement of input data set	proc sort data=names out=names;
no replacement of input data set	proc sort data=names out=namesbyid;

With all three replacement options (implicit replacement, explicit replacement, and no replacement) there must be at least enough space in the output library for a copy of the original data set.

You can also sort compressed data sets. If you specify a compressed data set as the input data set and omit the OUT= option, then the input data set is sorted and remains compressed. If you specify an OUT= data set, then the resulting data set is compressed only if you choose a compression method with the COMPRESS= data set option. For more information, see "COMPRESS= Data Set Option" in *SAS Viya Data Set Options: Reference* .

Also note that PROC SORT manipulates the uncompressed observation in memory and, if there is insufficient memory to complete the sort, stores the uncompressed data in a utility file. For these reasons, sorting compressed data sets might be intensive and require more storage than anticipated. Consider using the TAGSORT option when sorting compressed data sets.

Note: If the SAS system option NOREPLACE is in effect, then you cannot replace an original permanent data set with a sorted version. You must either use the OUT= option or specify the SAS system option REPLACE in an OPTIONS statement. The SAS system option NOREPLACE does not affect temporary SAS data sets.

Examples: SORT Procedure

Example 1: Sorting by the Values of Multiple Variables

Features: PROC SORT statement option
OUT=
BY statement

Other features: PROC PRINT

Details

This example does the following:

- sorts the observations by the values of two variables
- creates an output data set for the sorted observations
- prints the results

Program

```
data account;
   input Company $ 1-22 Debt 25-30 AccountNumber 33-36
         Town $ 39-51;
   datalines;
Paul's Pizza             83.00  1019  Apex
World Wide Electronics  119.95  1122  Garner
Strickland Industries   657.22  1675  Morrisville
Ice Cream Delight       299.98  2310  Holly Springs
Watson Tabor Travel      37.95  3131  Apex
```

```
Boyd & Sons Accounting  312.49  4762  Garner
Bob's Beds              119.95  4998  Morrisville
Tina's Pet Shop          37.95  5108  Apex
Elway Piano and Organ    65.79  5217  Garner
Tim's Burger Stand      119.95  6335  Holly Springs
Peter's Auto Parts       65.79  7288  Apex
Deluxe Hardware         467.12  8941  Garner
Pauline's Antiques      302.05  9112  Morrisville
Apex Catering            37.95  9923  Apex
;

proc sort data=account out=bytown;

   by town company;
run;

proc print data=bytown;

   var company town debt accountnumber;

   title  'Customers with Past-Due Accounts';
   title2 'Listed Alphabetically within Town';
run;
```

Program Description

Create the input data set ACCOUNT. ACCOUNT contains the name of each business that owes money, the amount of money that it owes on its account, the account number, and the town where the business is located.

```
data account;
   input Company $ 1-22 Debt 25-30 AccountNumber 33-36
         Town $ 39-51;
   datalines;
Paul's Pizza             83.00  1019  Apex
World Wide Electronics  119.95  1122  Garner
Strickland Industries   657.22  1675  Morrisville
Ice Cream Delight       299.98  2310  Holly Springs
Watson Tabor Travel      37.95  3131  Apex
Boyd & Sons Accounting  312.49  4762  Garner
Bob's Beds              119.95  4998  Morrisville
Tina's Pet Shop          37.95  5108  Apex
Elway Piano and Organ    65.79  5217  Garner
Tim's Burger Stand      119.95  6335  Holly Springs
Peter's Auto Parts       65.79  7288  Apex
Deluxe Hardware         467.12  8941  Garner
Pauline's Antiques      302.05  9112  Morrisville
Apex Catering            37.95  9923  Apex
;
```

Create the output data set BYTOWN. OUT= creates a new data set for the sorted observations.

```
proc sort data=account out=bytown;
```

Sort by two variables. The BY statement specifies that the observations should be first ordered alphabetically by town and then by company.

```
   by town company;
```

```
run;
```

Print the output data set BYTOWN. PROC PRINT prints the data set BYTOWN.

```
proc print data=bytown;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
   var company town debt accountnumber;
```

Specify the titles.

```
   title  'Customers with Past-Due Accounts';
   title2 'Listed Alphabetically within Town';
run;
```

Output: HTML

Output 18.3 *Sorting by the Values of Multiple Variables*

Customers with Past-Due Accounts
Listed Alphabetically within Town

Obs	Company	Town	Debt	AccountNumber
1	Apex Catering	Apex	37.95	9923
2	Paul's Pizza	Apex	83.00	1019
3	Peter's Auto Parts	Apex	65.79	7288
4	Tina's Pet Shop	Apex	37.95	5108
5	Watson Tabor Travel	Apex	37.95	3131
6	Boyd & Sons Accounting	Garner	312.49	4762
7	Deluxe Hardware	Garner	467.12	8941
8	Elway Piano and Organ	Garner	65.79	5217
9	World Wide Electronics	Garner	119.95	1122
10	Ice Cream Delight	Holly Springs	299.98	2310
11	Tim's Burger Stand	Holly Springs	119.95	6335
12	Bob's Beds	Morrisville	119.95	4998
13	Pauline's Antiques	Morrisville	302.05	9112
14	Strickland Industries	Morrisville	657.22	1675

Example 2: Sorting in Descending Order

Features: This example BY statement option
DESCENDING

Other features: PROC PRINT

Data set: Account

Details

This example does the following:

- sorts the observations by the values of three variables
- sorts one of the variables in descending order
- prints the results

Program

```
proc sort data=account out=sorted;

   by town descending debt accountnumber;
run;

proc print data=sorted;

   var company town debt accountnumber;

   title  'Customers with Past-Due Accounts';
   title2 'Listed by Town, Amount, Account Number';
run;
```

Program Description

Create the output data set SORTED. OUT= creates a new data set for the sorted observations.

```
proc sort data=account out=sorted;
```

Sort by three variables with one in descending order. The BY statement specifies that observations should be first ordered alphabetically by town, then by descending value of amount owed, then by ascending value of the account number.

```
   by town descending debt accountnumber;
run;
```

Print the output data set SORTED. PROC PRINT prints the data set SORTED.

```
proc print data=sorted;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
   var company town debt accountnumber;
```

Specify the titles.

```
   title  'Customers with Past-Due Accounts';
   title2 'Listed by Town, Amount, Account Number';
run;
```

Output: HTML

Note that sorting last by AccountNumber puts the businesses in Apex with a debt of $37.95 in order of account number.

Output 18.4 *Sorting in Descending Order*

Customers with Past-Due Accounts
Listed by Town, Amount, Account Number

Obs	Company	Town	Debt	AccountNumber
1	Paul's Pizza	Apex	83.00	1019
2	Peter's Auto Parts	Apex	65.79	7288
3	Watson Tabor Travel	Apex	37.95	3131
4	Tina's Pet Shop	Apex	37.95	5108
5	Apex Catering	Apex	37.95	9923
6	Deluxe Hardware	Garner	467.12	8941
7	Boyd & Sons Accounting	Garner	312.49	4762
8	World Wide Electronics	Garner	119.95	1122
9	Elway Piano and Organ	Garner	65.79	5217
10	Ice Cream Delight	Holly Springs	299.98	2310
11	Tim's Burger Stand	Holly Springs	119.95	6335
12	Strickland Industries	Morrisville	657.22	1675
13	Pauline's Antiques	Morrisville	302.05	9112
14	Bob's Beds	Morrisville	119.95	4998

Example 3: Creating a View with a Single BY Variable

The following example shows how to create a view by using a single BY variable. SAS uses the BEST argument in the SORTPGM system option to sort the data. By using BEST, SAS selects either the host sort or the SAS sort. (Sorting can also be performed by a DBMS when you use a SAS/ACCESS engine.)

```
options sortpgm=best msglevel=i;

data one;
   input name $ age;
   datalines;
Anne 35
ALBERT 10
JUAN 90
Janet 5
Bridget 23
BRIAN 45
```

```
;

data oneview / view=oneview;
   set one;
   name1=upcase(name);
run;

proc sort data=oneview out=final(drop=name1);
   by name1;
run;

proc print data=final;
run;
```

Log 18.2 *Log Output*

```
NOTE: SAS threaded sort was used.
```

Output 18.5 *Output from Creating a View with a Single BY Variable*

The SAS System

Obs	name	age
1	ALBERT	10
2	Anne	35
3	BRIAN	45
4	Bridget	23
5	Janet	5
6	JUAN	90

Example 4: Maintaining the Relative Order of Observations in Each BY Group

Features: PROC SORT statement option
EQUALS | NOEQUALS

Other features: PROC PRINT

Details

This example does the following:

- sorts the observations by the value of the first variable
- maintains the relative order with the EQUALS option
- does not maintain the relative order with the NOEQUALS option

Program

```
data insurance;
   input YearsWorked 1 InsuranceID 3-5;
   datalines;
5 421
5 336
1 209
1 564
3 711
3 343
4 212
4 616
;

proc sort data=insurance out=byyears1 equals;

   by yearsworked;
run;

proc print data=byyears1;

   var yearsworked insuranceid;

   title 'Sort with EQUALS';
run;

proc sort data=insurance out=byyears2 noequals;

   by yearsworked;
run;

proc print data=byyears2;

   var yearsworked insuranceid;

   title 'Sort with NOEQUALS';
run;
```

Program Description

Create the input data set INSURANCE. INSURANCE contains the number of years worked by all insured employees and their insurance IDs.

```
data insurance;
   input YearsWorked 1 InsuranceID 3-5;
   datalines;
5 421
5 336
1 209
1 564
3 711
3 343
4 212
4 616
;
```

Create the output data set BYYEARS1 with the EQUALS option. OUT= creates a new data set for the sorted observations. The EQUALS option maintains the order of the observations relative to each other.

```
proc sort data=insurance out=byyears1 equals;
```

Sort by the first variable. The BY statement specifies that the observations should be ordered numerically by the number of years worked.

```
   by yearsworked;
run;
```

Print the output data set BYYEARS1. PROC PRINT prints the data set BYYEARS1.

```
proc print data=byyears1;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
   var yearsworked insuranceid;
```

Specify the title.

```
   title 'Sort with EQUALS';
run;
```

Create the output data set BYYEARS2. OUT= creates a new data set for the sorted observations. The NOEQUALS option does not maintain the order of the observations relative to each other.

```
proc sort data=insurance out=byyears2 noequals;
```

Sort by the first variable. The BY statement specifies that the observations should be ordered numerically by the number of years worked.

```
   by yearsworked;
run;
```

Print the output data set BYYEARS2. PROC PRINT prints the data set BYYEARS2.

```
proc print data=byyears2;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
   var yearsworked insuranceid;
```

Specify the title.

```
   title 'Sort with NOEQUALS';
run;
```

Output: HTML

Note that sorting with the EQUALS option versus sorting with the NOEQUALS option causes a different sort order for the observations where YearsWorked=3.

Output 18.6 *Sorting with the EQUALS Option*

Sort with EQUALS

Obs	YearsWorked	InsuranceID
1	1	209
2	1	564
3	3	711
4	3	343
5	4	212
6	4	616
7	5	421
8	5	336

Output 18.7 *Sorting with the NOEQUALS Option*

Sort with NOEQUALS

Obs	YearsWorked	InsuranceID
1	1	209
2	1	564
3	3	343
4	3	711
5	4	212
6	4	616
7	5	421
8	5	336

Example 5: Retaining the First Observation of Each BY Group

Features: PROC SORT statement option
NODUPKEY
BY statement

Other features: PROC PRINT

Data set: Account

Note: The EQUALS option must be in effect to ensure that the first observation for each BY group is the one that is retained by the NODUPKEY option. The EQUALS option is the default. If the NOEQUALS option has been specified, then one observation for

each BY group is retained by the NODUPKEY option, but not necessarily the first observation.

Details

In this example, PROC SORT creates an output data set that contains only the first observation of each BY group. The NODUPKEY option prevents an observation from being written to the output data set when its BY value is identical to the BY value of the last observation written to the output data set. The resulting report contains one observation for each town where the businesses are located.

Program

```
proc sort data=account out=towns nodupkey;

   by town;
run;

proc print data=towns;

   var town company debt accountnumber;

   title 'Towns of Customers with Past-Due Accounts';
run;
```

Program Description

Create the output data set TOWNS but include only the first observation of each BY group. NODUPKEY writes only the first observation of each BY group to the new data set TOWNS. If you use the VMS operating environment sort, then the observation that is written to the output data set is not always the first observation of the BY group.

```
proc sort data=account out=towns nodupkey;
```

Sort by one variable. The BY statement specifies that observations should be ordered by town.

```
   by town;
run;
```

Print the output data set TOWNS. PROC PRINT prints the data set TOWNS.

```
proc print data=towns;
```

Specify the variables to be printed. The VAR statement specifies the variables to be printed and their column order in the output.

```
   var town company debt accountnumber;
```

Specify the title.

```
   title 'Towns of Customers with Past-Due Accounts';
run;
```

Output: HTML

The output data set contains only four observations, one for each town in the input data set.

Output 18.8 *Retaining the First Observation of Each BY Group*

Towns of Customers with Past-Due Accounts

Obs	Town	Company	Debt	AccountNumber
1	Apex	Paul's Pizza	83.00	1019
2	Garner	World Wide Electronics	119.95	1122
3	Holly Springs	Ice Cream Delight	299.98	2310
4	Morrisville	Strickland Industries	657.22	1675

Example 6: Linguistic Sorting Using ALTERNATE_HANDLING=

Features: PROC SORT statement option
sortseq=linguistic
ALTERNATE_HANDLING=SHIFTED
STRENGTH=3

BY statement

VAR statement

Other features: PROC PRINT

Note: For more information about strengthening the linguistic sort of strings, see "Example 7: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=" on page 419.

Details

In this example, PROC SORT creates an output data set that contains only the first observation of each BY group. You have specified ALTERNATE_HANDLING=SHIFTED because you want "a-b" to sort close to "ab" and "aB". That is, you do not want "a-b" to appear somewhere far away from "ab" and "aB" by virtue of its hyphen.

Note: In this example, the default STRENGTH for this locale is 3.

Notice how "a-b" and "ab" are treated equivalently in the following example. To order them beyond the first three levels of comparison (alphabetic, diacritic, and case), you can use the fourth level of comparison and specify STRENGTH=4. "Example 7: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=" on page 419 shows how to distinguish the strings further.

Program

```
data a;
   length x $ 10;
```

```
   x='a-b'; output;
   x='ab'; output;
   x='a-b'; output;
   x='aB'; output;
run;

proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED );
   by x;
run;

title1 "Linguistic Collation with ALTERNATE_HANDLING=SHIFTED";
proc print data=a;
run;

title1 "Linguistic Collation with ALTERNATE_HANDLING=SHIFTED and BY Processing";
proc print data=a;
   var x;
   by x;
run;
```

Program Description

Create the data set.

```
data a;
   length x $ 10;
   x='a-b'; output;
   x='ab'; output;
   x='a-b'; output;
   x='aB'; output;
run;
```

Sort the data set using linguistic sorting. Use linguistic sorting and the ALTERNATE_HANDLING=SHIFTED option to sort the data set. Note that the default STRENGTH for this locale is 3. Also use the BY statement to order observations by x.

```
proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED );
   by x;
run;
```

Print data set A. The TITLE1 statement tells the PRINT procedure the title to use for the output. PROC PRINT then prints data set A.

```
title1 "Linguistic Collation with ALTERNATE_HANDLING=SHIFTED";
proc print data=a;
run;
```

Print data set A using By processing. The TITLE1 statement tells the PRINT procedure the title to use for the output. PROC PRINT then prints data set A using By processing.

```
title1 "Linguistic Collation with ALTERNATE_HANDLING=SHIFTED and BY Processing";
proc print data=a;
   var x;
   by x;
run;
```

Output: HTML

The first PROC PRINT shows that the order of "a-b" and "ab" is not well defined. The second PROC PRINT uses BY processing to show that these values are considered equivalent. "Example 7: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=" on page 419 shows how to distinguish the strings more.

Output 18.9 *Linguistic Sorting Using the ALTERNATE_HANDLING Option*

Linguistic Collation with ALTERNATE_HANDLING=SHIFTED

Obs	x
1	a-b
2	ab
3	a-b
4	aB

Linguistic Collation with ALTERNATE_HANDLING=SHIFTED and BY Processing

x=a-b

Obs	x
1	a-b
2	ab
3	a-b

x=aB

Obs	x
4	aB

Example 7: Linguistic Sorting Using ALTERNATE_HANDLING= and STRENGTH=

Features: PROC SORT statement option
- sortseq=linguistic
- ALTERNATE_HANDLING=SHIFTED
- STRENGTH=4

BY statement

VAR statement

Other features: PROC PRINT

Details

In this example, PROC SORT creates an output data set that contains only the first observation of each BY group. In this example, ALTERNATE_HANDLING=SHIFTED is specified because you want "a-b" to sort close to "ab" and "aB" regardless of the hyphen.

Notice how "a-b" and "ab" are treated equivalently in the following example. However, if you want to further distinguish between them and have them appear in two separate BY groups, you must order the strings further. To order them beyond the first three

levels of comparison (alphabetic, diacritic, and case), use the fourth level of comparison, STRENGTH=4.

Program

```
data a;
   length x $ 10;
   x='a-b'; output;
   x='ab'; output;
   x='a-b'; output;
   x='aB'; output;
run;

proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED STRENGTH=4);
   by x;
run;

title1 "Linguistic Collation with STRENGTH=4";
proc print data=a;
run;

Title1 "Linguistic Collation with STRENGTH=4 and BY Processing";
proc print data=a;
   var x;
   by x;
run;
```

Program Description

Create the data set.

```
data a;
   length x $ 10;
   x='a-b'; output;
   x='ab'; output;
   x='a-b'; output;
   x='aB'; output;
run;
```

Sort the data set using linguistic sorting. Use linguistic sorting and the ALTERNATE_HANDLING=SHIFTED option to sort the data set. Note that the default STRENGTH for this locale is 4. The BY statement specifies that observations should be ordered by x.

```
proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED STRENGTH=4);
   by x;
run;
```

Print the output data set A. The TITLE1 statement tells the PRINT procedure the title to use for the output. PROC PRINT then prints data set A.

```
title1 "Linguistic Collation with STRENGTH=4";
proc print data=a;
run;
```

Print the output data set A using By processing. The TITLE statement tells the PRINT procedure what title to use for this output. PROC PRINT then prints data set A using By processing.

```
Title1 "Linguistic Collation with STRENGTH=4 and BY Processing";
proc print data=a;
   var x;
   by x;
run;
```

Output: HTML

The first PROC PRINT shows that the order of "a-b" and "ab" is not well defined. Differentiate between the two by setting STRENGTH=4. The second PROC PRINT uses BY processing to show the order of precedence and how they are differentiated.

Output 18.10 *Linguistic Sorting Using the ALTERNATE_HANDLING and STRENGTH Options*

Linguistic Collation with STRENGTH=4

Obs	x
1	a-b
2	a-b
3	ab
4	aB

Linguistic Collation with STRENGTH=4 and BY Processing

x=a-b

Obs	x
1	a-b
2	a-b

x=ab

Obs	x
3	ab

x=aB

Obs	x
4	aB

Chapter 19

TRANSPOSE Procedure

Overview: TRANSPOSE Procedure

What Does the TRANSPOSE Procedure Do?

The TRANSPOSE procedure creates an output data set by restructuring the values in a SAS data set, transposing selected variables into observations. The TRANSPOSE procedure can often eliminate the need to write a lengthy DATA step to achieve the same result. Further, the output data set can be used in subsequent DATA or PROC steps for analysis, reporting, or further data manipulation.

PROC TRANSPOSE does not produce printed output. To print the output data set from the PROC TRANSPOSE step, use PROC PRINT, PROC REPORT, or another SAS reporting tool.

To create a transposed variable, the procedure transposes the values of an observation in the input data set into values of a variable in the output data set.

What Types of Transpositions Can PROC TRANSPOSE Perform?

Simple Transposition

The following example illustrates a simple transposition. In the input data set, each variable represents the scores from one tester. In the output data set, each observation now represents the scores from one tester. Each value of _NAME_ is the name of a variable in the input data set that the procedure transposed. Thus, the value of _NAME_ identifies the source of each observation in the output data set. For example, the values in the first observation in the output data set come from the values of the variable Tester1 in the input data set. The statements that produce the output follow.

```
proc print data=proclib.product noobs;
   title 'The Input Data Set';
run;

proc transpose data=proclib.product
               out=proclib.product_transposed;
run;

proc print data=proclib.product_transposed noobs;
   title 'The Output Data Set';
run;
```

Output 19.1 *A Simple Transposition*

The Input Data Set 1

Tester1	Tester2	Tester3	Tester4
22	25	21	21
15	19	18	17
17	19	19	19
20	19	16	19
14	15	13	13
15	17	18	19
10	11	9	10
22	24	23	21

The Output Data Set 2

NAME	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8
Tester1	22	15	17	20	14	15	10	22
Tester2	25	19	19	19	15	17	11	24
Tester3	21	18	19	16	13	18	9	23
Tester4	21	17	19	19	13	19	10	21

Complex Transposition Using BY Groups

The next example, which uses BY groups, is more complex. The input data set represents measurements of the weight and length of fish at two lakes. The statements that create the output data set do the following:

- transpose only the variables that contain the length measurements
- create six BY groups, one for each lake and date
- use a data set option to name the transposed variable

Output 19.2 *A Transposition with BY Groups*

Input Data Set 1

Location	Date	Length1	Weight1	Length2	Weight2	Length3	Weight3	Length4	Weight4
Cole Pond	02JUN95	31	0.25	32	0.30	32	0.25	33	0.30
Cole Pond	03JUL95	33	0.32	34	0.41	37	0.48	32	0.28
Cole Pond	04AUG95	29	0.23	30	0.25	34	0.47	32	0.30
Eagle Lake	02JUN95	32	0.35	32	0.25	33	0.30	.	.
Eagle Lake	03JUL95	30	0.20	36	0.45	.	.	.	.
Eagle Lake	04AUG95	33	0.30	33	0.28	34	0.42	.	

Fish Length Data for Each Location and Date 2

Location	Date	_NAME_	Measurement
Cole Pond	02JUN95	Length1	31
Cole Pond	02JUN95	Length2	32
Cole Pond	02JUN95	Length3	32
Cole Pond	02JUN95	Length4	33
Cole Pond	03JUL95	Length1	33
Cole Pond	03JUL95	Length2	34
Cole Pond	03JUL95	Length3	37
Cole Pond	03JUL95	Length4	32
Cole Pond	04AUG95	Length1	29
Cole Pond	04AUG95	Length2	30
Cole Pond	04AUG95	Length3	34
Cole Pond	04AUG95	Length4	32
Eagle Lake	02JUN95	Length1	32
Eagle Lake	02JUN95	Length2	32
Eagle Lake	02JUN95	Length3	33
Eagle Lake	02JUN95	Length4	.
Eagle Lake	03JUL95	Length1	30
Eagle Lake	03JUL95	Length2	36
Eagle Lake	03JUL95	Length3	.
Eagle Lake	03JUL95	Length4	.
Eagle Lake	04AUG95	Length1	33
Eagle Lake	04AUG95	Length2	33
Eagle Lake	04AUG95	Length3	34
Eagle Lake	04AUG95	Length4	.

For a complete explanation of the SAS program that produces these results, see "Example 4: Transposing BY Groups" on page 440.

Syntax: TRANSPOSE Procedure

Restriction: If the DATA= and OUT= options point to CAS, the transpose is performed within CAS by invoking the CAS TRANSPOSE action.

Tips: Does not support the Output Delivery System

You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. You can also use any global statement. For a list, see "Global Statements" in *SAS Viya Statements: Reference*.

```
PROC TRANSPOSE <DATA=input-data-set> <DELIMITER=delimiter>
  <LABEL=label>
  <LET> <NAME=name> <OUT=output-data-set> <PREFIX=prefix>
  <SUFFIX=suffix>;
    BY <DESCENDING> variable-1
      <<DESCENDING> variable-2 …>
      <NOTSORTED>;
    COPY variable(s);
    ID variable;
        IDLABEL variable;
    VAR variable(s);
```

Statement	Task	Example
PROC TRANSPOSE	Create an output data set by restructuring the values in a SAS data set, transposing selected variables into observations	Ex. 1, Ex. 2, Ex. 3, Ex. 5
BY	Transpose each BY group	Ex. 4
COPY	Copy variables directly without transposing them	Ex. 6
ID	Specify a variable whose values name the transposed variables	Ex. 2
IDLABEL	Create labels for the transposed variables	Ex. 3
VAR	List the variables to transpose	Ex. 4, Ex. 6

PROC TRANSPOSE Statement

Creates an output data set by restructuring the values in a SAS data set, transposing selected variables into observations.

Tip: You can use data set options with the DATA= and OUT= options. You can also use any global statement. For a list, see "Global Statements" in *SAS Viya Statements: Reference*.

Examples:

"Example 1: Performing a Simple Transposition" on page 435
"Example 2: Naming Transposed Variables" on page 437
"Example 3: Labeling Transposed Variables" on page 438
"Example 4: Transposing BY Groups" on page 440
"Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values" on page 443
"Example 6: Transposing Data for Statistical Analysis" on page 445

Syntax

PROC TRANSPOSE <DATA=*input-data-set*> <DELIMITER=*delimiter*> <LABEL=*label*> <LET> <NAME=*name*> <OUT=*output-data-set*> <PREFIX=*prefix*> <SUFFIX=*suffix*>;

Optional Arguments

DATA= *input-data-set*
names the SAS data set to transpose.

Default most recently created SAS data set

DELIMITER= *delimiter*
specifies a delimiter to use in constructing names for transposed variables in the output data set. If specified, the delimiter is inserted between variable values if more than one variable has been specified in the ID statement.

Alias DELIM=

Tip You can use name literals (n-literals) for the value of DELIMITER. Name literals are helpful when specifying typographical or foreign characters, especially when VALIDVARNAME=ANY.

See "ID Statement" on page 431

LABEL= *label*
specifies a name for the variable in the output data set that contains the label of the variable that is being transposed to create the current observation.

Default _LABEL_

Tip You can use name literals (n-literals) for the value of LABEL. Name literals are helpful when specifying typographical or foreign characters, especially when VALIDVARNAME=ANY.

LET
allows duplicate values of an ID variable. PROC TRANSPOSE transposes the observation that contains the last occurrence of a particular ID value within the data set or BY group.

See "Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values" on page 443

NAME= *name*
specifies the name for the variable in the output data set that contains the name of the variable that is being transposed to create the current observation.

Default _NAME_

See "Example 2: Naming Transposed Variables" on page 437

OUT= ***output-data-set***
names the output data set. If *output-data-set* does not exist, then PROC TRANSPOSE creates it by using the DATA*n* naming convention.

Default DATA*n*

See "Example 1: Performing a Simple Transposition" on page 435

PREFIX= ***prefix***
specifies a prefix to use in constructing names for transposed variables in the output data set. For example, if PREFIX=VAR, then the names of the variables are VAR1, VAR2, …,VAR*n*.

Interaction When you use PREFIX= with an ID statement, the variable name begins with the prefix value followed by the ID value.

Tip You can use name literals (n-literals) for the value of PREFIX. Name literals are helpful when specifying typographical or foreign characters, especially when VALIDVARNAME=ANY.

See "Example 2: Naming Transposed Variables" on page 437

SUFFIX= ***suffix***
specifies a suffix to use in constructing names for transposed variables in the output data set.

Interaction When you use SUFFIX= with an ID statement, the value is appended to the ID value.

Tip You can use name literals (n-literals) for the value of SUFFIX. Name literals are helpful when specifying typographical or foreign characters, especially when VALIDVARNAME=ANY.

BY Statement

Defines BY groups.

Restriction: Do not use PROC TRANSPOSE with a BY statement or an ID statement if another user is updating the data set at the same time.

Example: "Example 4: Transposing BY Groups" on page 440

Syntax

BY <DESCENDING> *variable-1*
<<DESCENDING> *variable-1* …>
<NOTSORTED>;

Required Argument

variable

specifies the variable that PROC TRANSPOSE uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING

specifies that the data set is sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The data is grouped in another way, such as chronological order.

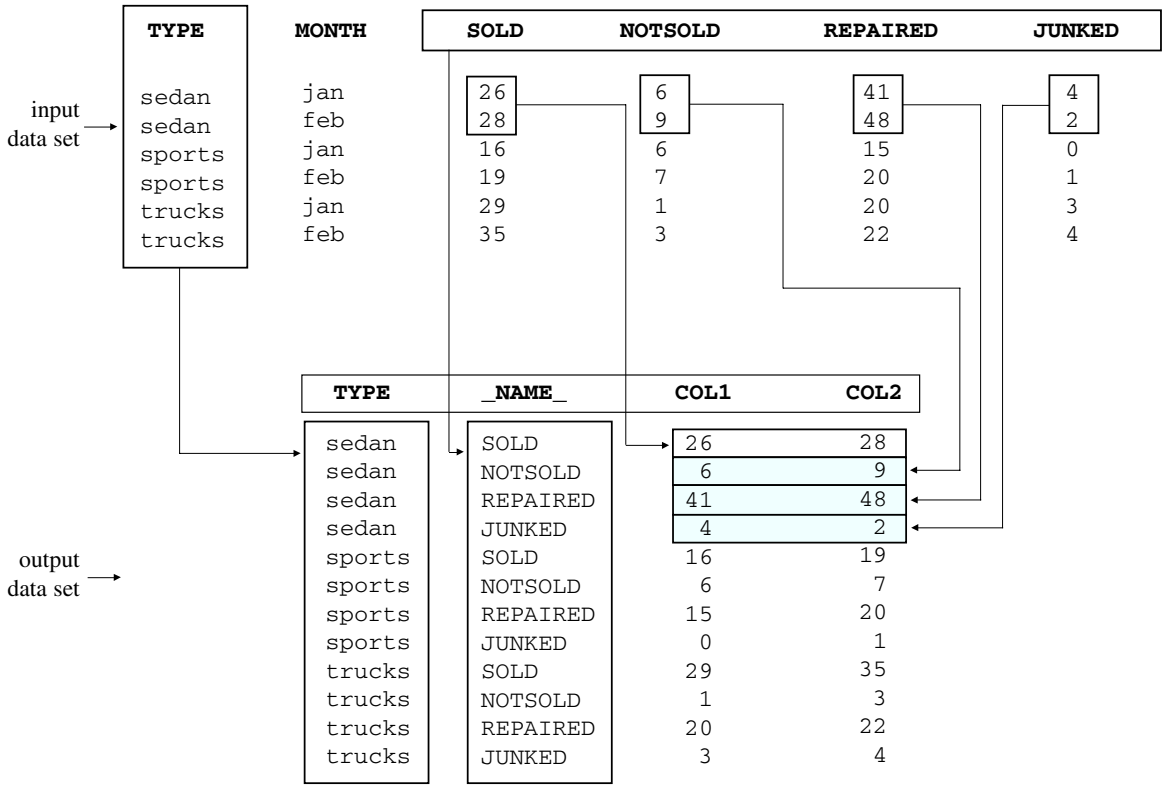
The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. The procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

The NOBYSORTED system option disables observation sequence checking system-wide and applies to all procedures and BY statements. See the "BYSORTED System Option" in *SAS Viya System Options: Reference*.

Details

The following figure shows what happens when you transpose a data set with BY groups. TYPE is the BY variable, and SOLD, NOTSOLD, REPAIRED, and JUNKED are the variables to transpose.

Figure 19.1 *Transposition with BY Groups*



- The number of observations in the output data set (12) is the number of BY groups (3) multiplied by the number of variables that are transposed (4).
- The BY variable is not transposed.
- _NAME_ contains the name of the variable in the input data set that was transposed to create the current observation in the output data set. You can use the NAME= option to specify another name for the _NAME_ variable.
- The maximum number of observations in any BY group in the input data set is two. Therefore, the output data set contains two variables, COL1 and COL2. COL1 and COL2 contain the values of SOLD, NOTSOLD, REPAIRED, and JUNKED.

Note: If a BY group in the input data set has more observations than other BY groups, then PROC TRANSPOSE assigns missing values in the output data set to the variables that have no corresponding input observations.

COPY Statement

Copies variables directly from the input data set to the output data set without transposing them.

Note: The COPY statement is not supported on the CAS server.

Example: "Example 6: Transposing Data for Statistical Analysis" on page 445

Syntax

COPY *variable(s)*;

Required Argument

variable(s)
names one or more variables that the COPY statement copies directly from the input data set to the output data set without transposing them.

Details

Because the COPY statement copies variables directly to the output data set, the number of observations in the output data set is equal to the number of observations in the input data set.

The procedure pads the output data set with missing values if the number of observations in the input data set is not equal to the number of variables that it transposes.

ID Statement

Specifies one or more variables in the input data set whose nonmissing formatted values name the transposed variables in the output data set. When a variable name is being formed in the transposed (output) data set, the formatted values of all listed ID variables are concatenated in the same order that the variables are listed in the ID statement. The PREFIX=, DELIMITER=, and SUFFIX= options can be used to modify the formed variable name. The PREFIX= option specifies a common character or character string to appear at the beginning of the formed variable names. The DELIMITER= option specifies a common character or character string to be inserted between the values of the ID variables. The SUFFIX= option specifies a common character or character string to be appended to the end of each formed variable name.

Restriction: You cannot use PROC TRANSPOSE with an ID statement or a BY statement with an engine that supports concurrent access if another user is updating the data set at the same time.

Tip: If the value of any ID variable is missing, then PROC TRANSPOSE writes a warning message to the log. The procedure does not transpose observations that have a missing value for any ID variable.

Example: "Example 2: Naming Transposed Variables" on page 437

Syntax

ID *variable(s)*;

Required Argument

variable(s)
names one or more variables whose formatted values are used to form the names of the variables in the output data set.

Details

Duplicate Output Data Set Variable Names

A variable name formed from the input data set ID variable values, combined with the PREFIX, DELIMITER, and SUFFIX option values, should be unique within the output

data set. An output data set variable name that occurs more than once indicates that two or more observations from the input data set are transposed to a single variable in the output data set and the result is data loss. This situation occurs when, in the case of a single ID variable, duplicate formatted values occur within the input data set or, if you use a BY statement, within a BY group. Similarly, this situation occurs in the case of multiple ID variables when the combination of formatted values of the ID variables occurs more than once within the input data set or BY group. To prevent data loss, if duplicate output data set variable names are formed, PROC TRANSPOSE issues an error message about duplicate ID values and stops processing. However, if the LET option is specified in the PROC TRANSPOSE statement then the procedure issues a warning message and continues processing, transposing the observation containing the last occurrence of the duplicate formatted variable values.

Note: The character substitutions and truncation required to ensure that the variable name formed from the ID variables is a valid SAS variable name, according to the VALIDVARNAME option setting, can result in duplicate output data set variable names even though the formatted value of the ID variable or combination of ID variables is unique within the input data set.

Making Variable Names Out of Numeric Values

When you use a numeric variable as an ID variable, PROC TRANSPOSE changes the formatted ID value into a valid SAS name.

SAS variable names cannot begin with a number unless you have set VALIDVARNAME=ANY. When the first character of the formatted value is numeric, the procedure prefixes an underscore to the value, this action truncates the last character of a 32-character value. Remaining invalid characters are replaced by underscores. The procedure truncates to 32 characters any ID value that is longer than 32 characters when the procedure uses that value to name a transposed variable.

If the formatted value looks like a numeric constant, then PROC TRANSPOSE changes the characters +, −, and . to P, N, and D, respectively. If the formatted value has characters that are not numeric, then PROC TRANSPOSE changes the characters +, −, and . to underscores.

Note: If the value of the VALIDVARNAME system option is V6, then PROC TRANSPOSE truncates transposed variable names to 8 characters.

Making Variable Names Out of Multiple ID Variables

When you specify a single ID variable, in forming an output data set variable name, the formatted values of the variable are made to conform with the SAS variable naming conventions imposed by the VALIDVARNAME option. The name formed by combining the ID variable values with the value of the PREFIX and SUFFIX options are also made to conform with the SAS variable naming convention. For both the formatted ID variable values and their combination with the PREFIX and SUFFIX options, invalid characters are replaced with underscores or, if the name appears to be a numeric constant, an underscore is used as a prefix and the characters +, -, and . are changed to P, N, and D. The resulting name is truncated to the maximum name length allowed by the VALIDVARNAME option setting. When you specify multiple ID variables, conformance to the SAS variable naming convention is imposed on the components of the variable name, using the formatted value of each ID variable, and also on the name composed from the ID variable values and the PREFIX, DELIMITER, and SUFFIX options. The resulting name is truncated to a length appropriate for the VALIDVARNAME option setting.

IDLABEL Statement

Creates labels for the transposed variables.

Restriction: Must appear after an ID statement.

Example: "Example 3: Labeling Transposed Variables" on page 438

Syntax

IDLABEL *variable*;

Required Argument

variable
names the variable whose values the procedure uses to label the variables that the ID statement names. *variable* can be character or numeric.

Note: To see the effect of the IDLABEL statement, print the output data set with the PRINT procedure by using the LABEL option. You can also print the contents of the output data set by using the CONTENTS statement in the DATASETS procedure.

VAR Statement

Lists the variables to transpose.

Examples: "Example 4: Transposing BY Groups" on page 440
"Example 6: Transposing Data for Statistical Analysis" on page 445

Syntax

VAR *variable(s)*;

Required Argument

variable(s)
names one or more variables to transpose.

Details

- If you omit the VAR statement, then the TRANSPOSE procedure transposes all numeric variables in the input data set that are not listed in another statement.
- You must list character variables in a VAR statement if you want to transpose them.

  Note: If the procedure is transposing any character variable, then all transposed variables are character variables.

Results: TRANSPOSE Procedure

Output Data Set

The TRANSPOSE procedure always produces an output data set, regardless of whether you specify the OUT= option in the PROC TRANSPOSE statement. PROC TRANSPOSE does not print the output data set. Use PROC PRINT, PROC REPORT, or some other SAS reporting tool to print the output data set.

Output Data Set Variables

The output data set contains the following variables:

- variables that result from transposing the values of each variable into an observation.
- a variable that PROC TRANSPOSE creates to identify the source of the values in each observation in the output data set. This variable is a character variable whose values are the names of the variables that are transposed from the input data set. By default, PROC TRANSPOSE names this variable _NAME_. To override the default name, use the NAME= option. The label for the _NAME_ variable is **NAME OF FORMER VARIABLE**.
- variables that PROC TRANSPOSE copies from the input data set when you use either the BY or COPY statement. These variables have the same names and values as they do in the input data set. These variables also have the same attributes (for example: type, length, label, informat, and format).
- a character variable whose values are the variable labels of the variables that are being transposed (if any of the variables that the procedure is transposing have labels). Specify the name of the variable by using the LABEL= option. The default is **_LABEL_**.

  Note: If the value of the LABEL= option or the NAME= option is the same as a variable that appears in a BY or COPY statement, then the output data set does not contain a variable whose values are the names or labels of the transposed variables.

Attributes of Transposed Variables

- All transposed variables are the same type and length.
- If all variables that the procedure is transposing are numeric, then the transposed variables are numeric. Thus, if the numeric variable has a character string as a formatted value, then its unformatted numeric value is transposed.
- If any variable that the procedure is transposing is character, then all transposed variables are character. If you are transposing a numeric variable that has a character string as a formatted value, then the formatted value is transposed.
- The length of the transposed variables is equal to the length of the longest variable that is being transposed.

Names of Transposed Variables

PROC TRANSPOSE uses the following rules to name transposed variables:

1. An ID statement specifies a variable or variables in the input data set whose formatted values become names for the transposed variables. If multiple ID variables are specified, the name of the transposed variable is the concatenation of the values of the ID variables. If the DELMITER= option is specified, its value is inserted between the formatted values of the ID variables when the names of the transposed variables are formed.
2. The PREFIX= option specifies a prefix to use in constructing the names of transposed variables. The SUFFIX= option also specifies a suffix to append to the names of the transposed variables.
3. If you do not use an ID statement, PREFIX= option, or the SUFFIX= option, then PROC TRANSPOSE looks for an input variable called _NAME_ to get the names of the transposed variables.
4. If you do not use an ID statement or the PREFIX= option, and if the input data set does not contain a variable named _NAME_, then PROC TRANSPOSE assigns the names COL1, COL2, …, COL*n* to the transposed variables.

Examples: TRANSPOSE Procedure

Example 1: Performing a Simple Transposition

Features: PROC TRANSPOSE statement option
OUT=

This example performs a default transposition and uses no subordinate statements.

Program

```
options nodate pageno=1 linesize=80 pagesize=40;

data score;
   input Student $9. +1 StudentID $ Section $ Test1 Test2 Final;
   datalines;
Capalleti 0545 1  94 91 87
Dubose    1252 2  51 65 91
Engles    1167 1  95 97 97
Grant     1230 2  63 75 80
Krupski   2527 2  80 76 71
Lundsford 4860 1  92 40 86
McBane    0674 1  75 78 72
;

proc transpose data=score out=score_transposed;
run;
```

```
proc print data=score_transposed noobs;
    title 'Student Test Scores in Variables';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create the SCORE data set. Set SCORE contains students' names, their identification numbers, and their grades on two tests and a final exam.

```
data score;
   input Student $9. +1 StudentID $ Section $ Test1 Test2 Final;
   datalines;
Capalleti 0545 1  94 91 87
Dubose    1252 2  51 65 91
Engles    1167 1  95 97 97
Grant     1230 2  63 75 80
Krupski   2527 2  80 76 71
Lundsford 4860 1  92 40 86
McBane    0674 1  75 78 72
;
```

Transpose the data set. PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears and none of the numeric variables appear in another statement. OUT= puts the result of the transposition in the data set SCORE_TRANSPOSED.

```
proc transpose data=score out=score_transposed;
run;
```

Print the SCORE_TRANSPOSED data set. The NOOBS option suppresses the printing of observation numbers

```
proc print data=score_transposed noobs;
    title 'Student Test Scores in Variables';
run;
```

Output

In the output data set SCORE_TRANSPOSED, the variables COL1 through COL7 contain the individual scores for the students. Each observation contains all the scores for one test. The variable _NAME_ contains the names of the variables from the input data set that were transposed.

Output 19.3 *Student Test Scores in Variables*

Student Test Scores in Variables

NAME	COL1	COL2	COL3	COL4	COL5	COL6	COL7
Test1	94	51	95	63	80	92	75
Test2	91	65	97	75	76	40	78
Final	87	91	97	80	71	86	72

Example 2: Naming Transposed Variables

Features: PROC TRANSPOSE statement options
NAME=
PREFIX=
ID statement

Data set: SCORE

This example uses the values of a variable and a user-supplied value to name transposed variables.

Program

```
options nodate pageno=1 linesize=80 pagesize=40;

proc transpose data=score out=idnumber name=Test
     prefix=sn;
    id studentid;
run;

proc print data=idnumber noobs;
   title 'Student Test Scores';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Transpose the data set. PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears. OUT= puts the result of the transposition in the IDNUMBER data set. NAME= specifies Test as the name for the variable that contains the names of the variables in the input data set that the procedure transposes. The procedure names the transposed variables by using the value from PREFIX=, sn, and the value of the ID variable StudentID.

```
proc transpose data=score out=idnumber name=Test
     prefix=sn;
    id studentid;
```

```
run;
```

Print the IDNUMBER data set. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=idnumber noobs;
   title 'Student Test Scores';
run;
```

Output

The following data set is the output data set, IDNUMBER.

Output 19.4 *Student Test Scores*

Student Test Scores

Test	sn0545	sn1252	sn1167	sn1230	sn2527	sn4860	sn0674
Test1	94	51	95	63	80	92	75
Test2	91	65	97	75	76	40	78
Final	87	91	97	80	71	86	72

Example 3: Labeling Transposed Variables

Features: PROC TRANSPOSE statement option
PREFIX=
IDLABEL statement

Data set: SCORE

This example uses the values of the variable in the IDLABEL statement to label transposed variables.

Program 1

```
options nodate pageno=1 linesize=80 pagesize=40;

proc transpose data=score out=idlabel name=Test
     prefix=sn;
   id studentid;

     idlabel student;
run;

  proc print data=idlabel label noobs;
     title 'Student Test Scores';
  run;

proc contents data=idlabel;
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Transpose the data set. PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears. OUT= puts the result of the transposition in the IDLABEL data set. NAME= specifies Test as the name for the variable that contains the names of the variables in the input data set that the procedure transposes. The procedure names the transposed variables by using the value from PREFIX=, sn, and the value of the ID variable StudentID.

```
proc transpose data=score out=idlabel name=Test
     prefix=sn;
   id studentid;
```

Assign labels to the output variables. PROC TRANSPOSE uses the values of the variable Student to label the transposed variables. The procedure provides the following as the label for the _NAME_ variable:NAME OF FORMER VARIABLE

```
     idlabel student;
run;
```

Print the IDLABEL data set. The LABEL option causes PROC PRINT to print variable labels for column headings. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=idlabel label noobs;
   title 'Student Test Scores';
run;
```

Display the IDLABEL variable names and label. PROC CONTENTS displays the variable names and labels.

```
proc contents data=idlabel;
run;
```

Output 1

This data set is the output data set, IDLABEL.

Output 19.5 *Student Test Scores, IDLABEL*

Student Test Scores

NAME OF FORMER VARIABLE	Capalleti	Dubose	Engles	Grant	Krupski	Lundsford	McBane
Test1	94	51	95	63	80	92	75
Test2	91	65	97	75	76	40	78
Final	87	91	97	80	71	86	72

Program 2

```
proc contents data=idlabel;
run;
```

Program Description

Display the variable and label names. PROC CONTENTS displays the variable names and the labels used in the first program.

```
proc contents data=idlabel;
run;
```

Output 2

In the following output, PROC CONTENTS displays the variables and labels.

Output 19.6 *The Contents Procedure*

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Label
1	Test	Char	8	NAME OF FORMER VARIABLE
2	sn0545	Num	8	Capalleti
8	sn0674	Num	8	McBane
4	sn1167	Num	8	Engles
5	sn1230	Num	8	Grant
3	sn1252	Num	8	Dubose
6	sn2527	Num	8	Krupski
7	sn4860	Num	8	Lundsford

Example 4: Transposing BY Groups

Features: BY statement
VAR statement

Other features: Data set option
RENAME=

This example illustrates transposing BY groups and selecting variables to transpose.

Program

```
options nodate pageno=1 linesize=80 pagesize=40;

data fishdata;
   infile datalines missover;
   input Location & $10. Date date7.
         Length1 Weight1 Length2 Weight2 Length3 Weight3
         Length4 Weight4;
   format date date7.;
```

```
   datalines;
Cole Pond   2JUN95 31 .25 32 .3  32 .25 33 .3
Cole Pond   3JUL95 33 .32 34 .41 37 .48 32 .28
Cole Pond   4AUG95 29 .23 30 .25 34 .47 32 .3
Eagle Lake  2JUN95 32 .35 32 .25 33 .30
Eagle Lake  3JUL95 30 .20 36 .45
Eagle Lake  4AUG95 33 .30 33 .28 34 .42
;

proc transpose data=fishdata
     out=fishlength(rename=(col1=Measurement));

   var length1-length4;

   by location date;
run;

proc print data=fishlength noobs;
   title 'Fish Length Data for Each Location and Date';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create the FISHDATA data set. The data in FISHDATA represents length and weight measurements of fish that were caught at two ponds on three separate days. The raw data is sorted by Location and Date.

```
data fishdata;
   infile datalines missover;
   input Location & $10. Date date7.
         Length1 Weight1 Length2 Weight2 Length3 Weight3
         Length4 Weight4;
   format date date7.;
   datalines;
Cole Pond   2JUN95 31 .25 32 .3  32 .25 33 .3
Cole Pond   3JUL95 33 .32 34 .41 37 .48 32 .28
Cole Pond   4AUG95 29 .23 30 .25 34 .47 32 .3
Eagle Lake  2JUN95 32 .35 32 .25 33 .30
Eagle Lake  3JUL95 30 .20 36 .45
Eagle Lake  4AUG95 33 .30 33 .28 34 .42
;
```

Transpose the data set. OUT= puts the result of the transposition in the FISHLENGTH data set. RENAME= renames COL1 in the output data set to Measurement.

```
proc transpose data=fishdata
     out=fishlength(rename=(col1=Measurement));
```

Specify the variables to transpose. The VAR statement limits the variables that PROC TRANSPOSE transposes.

```
   var length1-length4;
```

Organize the output data set into BY groups. The BY statement creates BY groups for each unique combination of values of Location and Date. The procedure does not transpose the BY variables.

```
   by location date;
run;
```

Print the FISHLENGTH data set. The NOOBS option suppresses the printing of observation numbers.

```
proc print data=fishlength noobs;
   title 'Fish Length Data for Each Location and Date';
run;
```

Output

The following data set is the output data set, FISHLENGTH. For each BY group in the original data set, PROC TRANSPOSE creates four observations, one for each variable that it is transposing. Missing values appear for the variable Measurement (renamed from COL1) when the variables that are being transposed have no value in the input data set for that BY group. Several observations have a missing value for Measurement. For example, in the last observation, a missing value appears because the input data contained no value for Length4 on 04AUG95 at Eagle Lake.

Output 19.7 *Fish Length Data*

Fish Length Data for Each Location and Date

Location	Date	_NAME_	Measurement
Cole Pond	02JUN95	Length1	31
Cole Pond	02JUN95	Length2	32
Cole Pond	02JUN95	Length3	32
Cole Pond	02JUN95	Length4	33
Cole Pond	03JUL95	Length1	33
Cole Pond	03JUL95	Length2	34
Cole Pond	03JUL95	Length3	37
Cole Pond	03JUL95	Length4	32
Cole Pond	04AUG95	Length1	29
Cole Pond	04AUG95	Length2	30
Cole Pond	04AUG95	Length3	34
Cole Pond	04AUG95	Length4	32
Eagle Lake	02JUN95	Length1	32
Eagle Lake	02JUN95	Length2	32
Eagle Lake	02JUN95	Length3	33
Eagle Lake	02JUN95	Length4	.
Eagle Lake	03JUL95	Length1	30
Eagle Lake	03JUL95	Length2	36
Eagle Lake	03JUL95	Length3	.
Eagle Lake	03JUL95	Length4	.
Eagle Lake	04AUG95	Length1	33
Eagle Lake	04AUG95	Length2	33
Eagle Lake	04AUG95	Length3	34
Eagle Lake	04AUG95	Length4	.

Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values

Features: PROC TRANSPOSE statement option
LET

This example shows how to use values of a variable (ID) to name transposed variables even when the ID variable has duplicate values.

Program

```
options nodate pageno=1 linesize=64 pagesize=40;

data stocks;
    input Company $14. Date $ Time $ Price;
```

```
    datalines;
Horizon Kites jun11 opening 29
Horizon Kites jun11 noon    27
Horizon Kites jun11 closing 27
Horizon Kites jun12 opening 27
Horizon Kites jun12 noon    28
Horizon Kites jun12 closing 30
SkyHi Kites   jun11 opening 43
SkyHi Kites   jun11 noon    43
SkyHi Kites   jun11 closing 44
SkyHi Kites   jun12 opening 44
SkyHi Kites   jun12 noon    45
SkyHi Kites   jun12 closing 45
;

proc transpose data=stocks out=close let;

   by company;

   id date;
 run;

proc print data=close noobs;
   title 'Closing Prices for Horizon Kites and SkyHi Kites';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=64 pagesize=40;
```

Create the STOCKS data set. STOCKS contains stock prices for two competing kite manufacturers. The prices are recorded for two days, three times a day: at opening, at noon, and at closing. Notice that the input data set contains duplicate values for the Date variable.

```
data stocks;
    input Company $14. Date $ Time $ Price;
    datalines;
Horizon Kites jun11 opening 29
Horizon Kites jun11 noon    27
Horizon Kites jun11 closing 27
Horizon Kites jun12 opening 27
Horizon Kites jun12 noon    28
Horizon Kites jun12 closing 30
SkyHi Kites   jun11 opening 43
SkyHi Kites   jun11 noon    43
SkyHi Kites   jun11 closing 44
SkyHi Kites   jun12 opening 44
SkyHi Kites   jun12 noon    45
SkyHi Kites   jun12 closing 45
;
```

Transpose the data set. LET transposes only the last observation for each BY group. PROC TRANSPOSE transposes only the Price variable. OUT= puts the result of the transposition in the CLOSE data set.

```
proc transpose data=stocks out=close let;
```

Organize the output data set into BY groups. The BY statement creates two BY groups, one for each company.

```
   by company;
```

Name the transposed variables. The values of Date are used as names for the transposed variables.

```
   id date;
 run;
```

Print the CLOSE data set. The NOOBS option suppresses the printing of observation numbers..

```
proc print data=close noobs;
   title 'Closing Prices for Horizon Kites and SkyHi Kites';
run;
```

Output

The following data set is the output data set, CLOSE.

Output 19.8 *Closing Prices*

Closing Prices for Horizon Kites and SkyHi Kites

Company	_NAME_	jun11	jun12
Horizon Kites	Price	27	30
SkyHi Kites	Price	44	45

Example 6: Transposing Data for Statistical Analysis

Features: COPY statement
VAR statement

This example arranges data to make it suitable for either a multivariate or a univariate repeated-measures analysis.

The data is from Chapter 8, “Repeated-Measures Analysis of Variance,” in SAS System for Linear Models, Third Edition.

Program 1

```
options nodate pageno=1 linesize=80 pagesize=40;

data weights;
   input Program $ s1-s7;
   datalines;
```

```
CONT  85 85 86 85 87 86 87
CONT  80 79 79 78 78 79 78
CONT  78 77 77 77 76 76 77
CONT  84 84 85 84 83 84 85
CONT  80 81 80 80 79 79 80
RI    79 79 79 80 80 78 80
RI    83 83 85 85 86 87 87
RI    81 83 82 82 83 83 82
RI    81 81 81 82 82 83 81
RI    80 81 82 82 82 84 86
WI    84 85 84 83 83 83 84
WI    74 75 75 76 75 76 76
WI    83 84 82 81 83 83 82
WI    86 87 87 87 87 87 86
WI    82 83 84 85 84 85 86
;

data split;
   set weights;
   array s{7} s1-s7;
   Subject + 1;
   do Time=1 to 7;
      Strength=s{time};
      output;
   end;
   drop s1-s7;
run;

proc print data=split(obs=15) noobs;
   title 'SPLIT Data Set';
   title2 'First 15 Observations Only';
run;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create the WEIGHTS data set. The data in WEIGHTS represents the results of an exercise therapy study of three weight-lifting programs: CONT is a control group, RI is a program in which the number of repetitions is increased, and WI is a program in which the weight is increased.

```
data weights;
   input Program $ s1-s7;
   datalines;
CONT  85 85 86 85 87 86 87
CONT  80 79 79 78 78 79 78
CONT  78 77 77 77 76 76 77
CONT  84 84 85 84 83 84 85
CONT  80 81 80 80 79 79 80
RI    79 79 79 80 80 78 80
RI    83 83 85 85 86 87 87
```

```
RI    81 83 82 82 83 83 82
RI    81 81 81 82 82 83 81
RI    80 81 82 82 82 84 86
WI    84 85 84 83 83 83 84
WI    74 75 75 76 75 76 76
WI    83 84 82 81 83 83 82
WI    86 87 87 87 87 87 86
WI    82 83 84 85 84 85 86
;
```

Create the SPLIT data set. This DATA step rearranges WEIGHTS to create the data set SPLIT. The DATA step transposes the strength values and creates two new variables: Time and Subject. SPLIT contains one observation for each repeated measure. SPLIT can be used in a PROC GLM step for a univariate repeated-measures analysis.

```
data split;
   set weights;
   array s{7} s1-s7;
   Subject + 1;
   do Time=1 to 7;
      Strength=s{time};
      output;
   end;
   drop s1-s7;
run;
```

Print the SPLIT data set. The NOOBS options suppresses the printing of observation numbers. The OBS= data set option limits the printing to the first 15 observations. SPLIT has 105 observations.

```
proc print data=split(obs=15) noobs;
   title 'SPLIT Data Set';
   title2 'First 15 Observations Only';
run;
```

Output 1

Output 19.9 *Split Data Set*

SPLIT Data Set
First 15 Observations Only

Program	Subject	Time	Strength
CONT	1	1	85
CONT	1	2	85
CONT	1	3	86
CONT	1	4	85
CONT	1	5	87
CONT	1	6	86
CONT	1	7	87
CONT	2	1	80
CONT	2	2	79
CONT	2	3	79
CONT	2	4	78
CONT	2	5	78
CONT	2	6	79
CONT	2	7	78
CONT	3	1	78

Program 2

```
options nodate pageno=1 linesize=80 pagesize=40;

proc transpose data=split out=totsplit prefix=Str;

   by program subject;
   copy time strength;

   var strength;
run;

proc print data=totsplit(obs=15) noobs;
   title  'TOTSPLIT Data Set';
   title2 'First 15 Observations Only';
run;
```

Program Description

Set the SAS system options.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Transpose the SPLIT data set. PROC TRANSPOSE transposes SPLIT to create TOTSPLIT. The TOTSPLIT data set contains the same variables as SPLIT and a variable for each strength measurement (Str1-Str7). TOTSPLIT can be used for either a multivariate repeated-measures analysis or a univariate repeated-measures analysis.

```
proc transpose data=split out=totsplit prefix=Str;
```

Organize the output data set into BY groups, and populate each BY group with untransposed values. The variables in the BY and COPY statements are not transposed. TOTSPLIT contains the variables Program, Subject, Time, and Strength with the same values that are in SPLIT. The BY statement creates the first observation in each BY group, which contains the transposed values of Strength. The COPY statement creates the other observations in each BY group by copying the values of Time and Strength without transposing them.

```
   by program subject;
   copy time strength;
```

Specify the variable to transpose. The VAR statement specifies the Strength variable as the only variable to be transposed.

```
   var strength;
run;
```

Print the TOTSPLIT data set. The NOOBS options suppresses the printing of observation numbers. The OBS= data set option limits the printing to the first 15 observations. SPLIT has 105 observations.

```
proc print data=totsplit(obs=15) noobs;
   title  'TOTSPLIT Data Set';
   title2 'First 15 Observations Only';
run;
```

Output 2

In the following output, the variables in TOTSPLIT with missing values are used only in a multivariate repeated-measures analysis. The missing values do not preclude this data set from being used in a repeated-measures analysis because the MODEL statement in PROC GLM ignores observations with missing values.

Output 19.10 *TOTSPLIT Data Set*

TOTSPLIT Data Set
First 15 Observations Only

Program	Subject	Time	Strength	_NAME_	Str1	Str2	Str3	Str4	Str5	Str6	Str7
CONT	1	1	85	Strength	85	85	86	85	87	86	87
CONT	1	2	85		.	.	.	.	.	.	.
CONT	1	3	86		.	.	.	.	.	.	.
CONT	1	4	85		.	.	.	.	.	.	.
CONT	1	5	87		.	.	.	.	.	.	.
CONT	1	6	86		.	.	.	.	.	.	.
CONT	1	7	87		.	.	.	.	.	.	.
CONT	2	1	80	Strength	80	79	79	78	78	79	78
CONT	2	2	79		.	.	.	.	.	.	.
CONT	2	3	79		.	.	.	.	.	.	.
CONT	2	4	78		.	.	.	.	.	.	.
CONT	2	5	78		.	.	.	.	.	.	.
CONT	2	6	79		.	.	.	.	.	.	.
CONT	2	7	78		.	.	.	.	.	.	.
CONT	3	1	78	Strength	78	77	77	77	76	76	77

Part 2

Appendixes

Appendix 1

Raw Data and DATA Steps for Base SAS Procedures

Overview of Raw Data and DATA Steps for Base SAS Procedures

The following raw data examples and DATA step examples are for use with the Base SAS Procedures.

The programs for examples in this document generally show you how to create the data sets that are used. Some examples show only partial data. For these examples, the complete data is shown in this appendix.

AddressPrint

```
James:Bunsen:6649 N Blue Gum St:New Orleans:LA:504-621-8927:jbunsen@gmail.com:
Josephine:Darakjy:4 B Blue Ridge Blvd:Brighton:MI:810-292-9388:josephine_darakjy@darakjy
Asuncion:Venere:8 W Cerritos Ave #54:Bridgeport:NJ:856-636-8749:asuncion@venere.org:
Lenna:Paprocki:639 Main St:Anchorage:AK:907-385-4412:lpaprocki@hotmail.com:
Donette:Foller:34 Center St:Hamilton:OH:513-570-1893:donette.foller@cox.net:
Simona:Morasca:3 Mcauley Dr:Ashland:OH:419-503-2484:simona@morasca.com:
Mitsue:Tollner:7 Eads St:Chicago:IL:773-573-6914:mitsue_tollner@yahoo.com:
Leota:Dilliard:7 W Jackson Blvd:San Jose:CA:408-752-3500:leota@hotmail.com:
Sage:Wieser:5 Boston Ave #88:Sioux Falls:SD:605-414-2147:sage_wieser@cox.net:
Kris:Marrier:228 Runamuck Pl #2808:Baltimore:MD:410-655-8723:kris@gmail.com:
Minna:Amigon:2371 Jerrold Ave:Kulpsville:PA:215-874-1229:minna_amigon@yahoo.com:
Adelie:Maclead:37275 St  Rt 17m M:Middle Island:NY:631-335-3414:amaclead@gmail.com:
Kiley:Caldarera:25 E 75th St #69:Los Angeles:CA:310-498-5651:kiley.caldarera@aol.com:
Graciela:Ruta:98 Connecticut Ave Nw:Chagrin Falls:OH:440-780-8425:gruta@cox.net:
Cammy:Albares:56 E Morehead St:Laredo:TX:956-537-6195:calbares@gmail.com:
Mattie:Poquette:73 State Road 434 E:Phoenix:AZ:602-277-4385:mattie@aol.com:
Meaghan:Garufi:69734 E Carrillo St:Mc Minnville:TN:931-313-9635:meaghan@hotmail.com:
Gladys:Rim:322 New Horizon Blvd:Milwaukee:WI:414-661-9598:gladys.rim@rim.org:
Yuki:Whobrey:1 State Route 27:Taylor:MI:313-288-7937:yuki_whobrey@aol.com:
Fletcher:Flosi:394 Manchester Blvd:Rockford:IL:815-828-2147:fletcher.flosi@yahoo.com:
Bette:Nicka:6 S 33rd St:Aston:PA:610-545-3615:bette_nicka@cox.net:
Veronika:Inouye:6 Greenleaf Ave:San Jose:CA:408-540-1785:vinouye@aol.com:
Willard:Kolmetz:618 W Yakima Ave:Irving:TX:972-303-9197:willard@hotmail.com:
Maryann:Royster:74 S Westgate St:Albany:NY:518-966-7987:mroyster@royster.com:
Alisha:Slusarski:3273 State St:Middlesex:NJ:732-658-3154:alisha@slusarski.com:
Allene:Iturbide:1 Central Ave:Stevens Point:WI:715-662-6764:allene_iturbide@cox.net:
Chanel:Caudy:86 Nw 66th St #8673:Shawnee:KS:913-388-2079:chanel.caudy@caudy.org:
Ezekiel:Chui:2 Cedar Ave #84:Easton:MD:410-669-1642:ezekiel@chui.com:
Willow:Kusko:90991 Thorburn Ave:New York:NY:212-582-4976:wkusko@yahoo.com:
Bernardo:Figeroa:386 9th Ave N:Conroe:TX:936-336-3951:bfigeroa@aol.com:
Ammie:Corrio:74874 Atlantic Ave:Columbus:OH:614-801-9788:ammie@corrio.com:
Francine:Vocelka:366 South Dr:Las Cruces:NM:505-977-3911:francine_vocelka@vocelka.com:
Ernie:Stenseth:45 E Liberty St:Ridgefield Park:NJ:201-709-6245:ernie_stenseth@aol.com:
Albina:Glick:4 Ralph Ct:Dunellen:NJ:732-924-7882:albina@glick.com:
Alishia:Sergi:2742 Distribution Way:New York:NY:212-860-1579:asergi@gmail.com:
Solange:Shinko:426 Wolf St:Metairie:LA:504-979-9175:solange@shinko.com:
Jose:Stockham:128 Bransten Rd:New York:NY:212-675-8570:jose@yahoo.com:
Rozella:Ostrosky:17 Morena Blvd:Camarillo:CA:805-832-6163:rozella.ostrosky@ostrosky.com:
```

```
Valentine:Gillian:775 W 17th St:San Antonio:TX:210-812-9597:valentine_gillian@gmail.com:
Kati:Rulapaugh:6980 Dorsett Rd:Abilene:KS:785-463-7829:kati.rulapaugh@hotmail.com:
Youlanda:Schemmer:2881 Lewis Rd:Prineville:OR:541-548-8197:youlanda@aol.com:
Dyan:Oldroyd:7219 Woodfield Rd:Overland Park:KS:913-413-4604:doldroyd@aol.com:
Roxane:Campain:1048 Main St:Fairbanks:AK:907-231-4722:roxane@hotmail.com:
Lavera:Perin:678 3rd Ave:Miami:FL:305-606-7291:lperin@perin.org:
Erick:Ferencz:20 S Babcock St:Fairbanks:AK:907-741-1044:erick.ferencz@aol.com:
Fatima:Saylors:2 Lighthouse Ave:Hopkins:MN:952-768-2416:fsaylors@saylors.org:
Jina:Briddick:38938 Park Blvd:Boston:MA:617-399-5124:jina_briddick@briddick.com:
Kanisha:Waycott:5 Tomahawk Dr:Los Angeles:CA:323-453-2780:kanisha_waycott@yahoo.com:
Emerson:Bowley:762 S Main St:Madison:WI:608-336-7444:emerson.bowley@bowley.org:
```

CARSURVEY

```
data carsurvey;
   input Rater Age Progressa Remark Jupiter Dynamo;
   datalines;
1   38  94  98  84  80
2   49  96  84  80  77
3   16  64  78  76  73
4   27  89  73  90  92
5   50  93  79  84  34
6   57  92  89  75  89
7   21  88  90  89  91
8   39  88  87  76  64
9   26  77  94  93  47
10  17  68  72  85  79
11  38  94  93  84  70
12  29  78  97  74  33
13  41  89  83  75  82
14  37  54  98  70  83
15  52  92  85  88  78
16  23  85  89  89  95
17  61  92  88  77  85
18  24  87  88  88  87
19  18  54  50  62  74
20  62  90  91  90  86
21  49  94  98  84  80
22  16  96  84  80  77
23  27  64  78  76  73
24  50  89  73  90  92
25  57  93  79  84  34
26  21  92  86  75  93
27  39  88  97  89  91
28  26  88  87  76  64
29  17  77  94  93  47
30  38  68  72  85  79
31  29  94  93  84  70
32  41  78  97  74  33
33  37  89  83  75  82
34  52  54  98  70  83
```

```
35  23  92  85  88  78
36  61  85  93  89  66
37  24  92  88  77  85
38  18  87  88  88  87
39  62  54  50  62  74
40  38  90  91  90  86
41  57  94  98  84  80
42  16  96  84  80  77
43  19  64  78  76  73
44  59  89  73  90  92
45  57  93  79  84  34
46  21  92  86  75  90
47  39  88  97  89  91
48  26  88  87  76  64
49  17  77  94  93  47
50  56  68  72  85  79
51  29  94  93  84  70
52  41  78  97  74  33
53  37  89  83  75  82
54  52  54  98  70  83
55  17  92  85  88  78
56  61  85  93  89  66
57  24  92  85  77  85
58  18  87  88  88  87
59  62  54  50  62  74
60  38  90  91  90  86
61  38  94  98  84  80
62  49  96  84  80  77
63  16  64  78  76  73
64  27  89  73  90  92
65  50  93  79  84  34
66  57  92  89  75  89
67  21  88  93  89  91
68  39  88  87  76  64
69  26  77  94  93  47
70  17  68  72  85  79
71  38  94  93  84  70
72  29  78  97  74  33
73  41  89  83  75  82
74  37  54  98  70  83
75  52  92  85  88  78
76  23  85  93  89  88
77  61  92  88  77  85
78  24  87  88  88  91
79  18  54  50  62  74
80  62  90  91  90  86
;
```

CENSUS

```
data census;
   input Density CrimeRate State $ 14-27 PostalCode $ 29-30;
   datalines;
```

```
263.3 4575.3 Ohio           OH
62.1 7017.1  Washington     WA
103.4 5161.9 South Carolina SC
53.4 3438.6  Mississippi    MS
180.0 8503.2 Florida        FL
80.8 2190.7  West Virginia  WV
428.7 5477.6 Maryland       MD
71.2 4707.5  Missouri       MO
43.9 4245.2  Arkansas       AR
7.3 6371.4   Nevada         NV
264.3 3163.2 Pennsylvania   PA
11.5 4156.3  Idaho          ID
44.1 6025.6  Oklahoma       OK
51.2 4615.8  Minnesota      MN
55.2 4271.2  Vermont        VT
27.4 6969.9  Oregon         OR
205.3 5416.5 Illinois       IL
94.1 5792.0  Georgia        GA
9.1 2678.0   South Dakota   SD
9.4 2833.0   North Dakota   ND
102.4 3371.7 New Hampshire  NH
54.3 7722.4  Texas          TX
76.6 4451.4  Alabama        AL
307.6 4938.8 Delaware       DE
151.4 6506.4 California     CA
111.6 4665.6 Tennessee      TN
120.4 4649.9 North Carolina NC
;
```

CHARITY

```
data Charity;
   input School $ 1-7 Year 9-12 Name $ 14-20 MoneyRaised 22-26
         HoursVolunteered 28-29;
   datalines;
Monroe  2007 Allison 31.65 19
Monroe  2007 Barry   23.76 16
Monroe  2007 Candace 21.11  5
Monroe  2007 Danny    6.89 23
Monroe  2007 Edward  53.76 31
Monroe  2007 Fiona   48.55 13
Monroe  2007 Gert    24.00 16
Monroe  2007 Harold  27.55 17
Monroe  2007 Ima     15.98  9
Monroe  2007 Jack    20.00 23
Monroe  2007 Katie   22.11  2
Monroe  2007 Lisa    18.34 17
Monroe  2007 Tonya   55.16 40
Monroe  2007 Max     26.77 34
Monroe  2007 Ned     28.43 22
Monroe  2007 Opal    32.66 14
Monroe  2008 Patsy   18.33 18
Monroe  2008 Quentin 16.89 15
```

```
Monroe  2008 Randall 12.98 17
Monroe  2008 Sam     15.88  5
Monroe  2008 Tyra    21.88 23
Monroe  2008 Myrtle  47.33 26
Monroe  2008 Frank   41.11 22
Monroe  2008 Cameron 65.44 14
Monroe  2008 Vern    17.89 11
Monroe  2008 Wendell 23.00 10
Monroe  2008 Bob     26.88  6
Monroe  2008 Leah    28.99 23
Monroe  2009 Becky   30.33 26
Monroe  2009 Sally   35.75 27
Monroe  2009 Edgar   27.11 12
Monroe  2009 Dawson  17.24 16
Monroe  2009 Lou      5.12 16
Monroe  2009 Damien  18.74 17
Monroe  2009 Mona    27.43  7
Monroe  2009 Della   56.78 15
Monroe  2009 Monique 29.88 19
Monroe  2009 Carl    31.12 25
Monroe  2009 Reba    35.16 22
Monroe  2009 Dax     27.65 23
Monroe  2009 Gary    23.11 15
Monroe  2009 Suzie   26.65 11
Monroe  2009 Benito  47.44 18
Monroe  2009 Thomas  21.99 23
Monroe  2009 Annie   24.99 27
Monroe  2009 Paul    27.98 22
Monroe  2009 Alex    24.00 16
Monroe  2009 Lauren  15.00 17
Monroe  2009 Julia   12.98 15
Monroe  2009 Keith   11.89 19
Monroe  2009 Jackie  26.88 22
Monroe  2009 Pablo   13.98 28
Monroe  2009 L.T.    56.87 33
Monroe  2009 Willard 78.65 24
Monroe  2009 Kathy   32.88 11
Monroe  2009 Abby    35.88 10
Kennedy 2007 Arturo  34.98 14
Kennedy 2007 Grace   27.55 25
Kennedy 2007 Winston 23.88 22
Kennedy 2007 Vince   12.88 21
Kennedy 2007 Claude  15.62  5
Kennedy 2007 Mary    28.99 34
Kennedy 2007 Abner   25.89 22
Kennedy 2007 Jay     35.89 35
Kennedy 2007 Alicia  28.77 26
Kennedy 2007 Freddy  29.00 27
Kennedy 2007 Eloise  31.67 25
Kennedy 2007 Jenny   43.89 22
Kennedy 2007 Thelma  52.63 21
Kennedy 2007 Tina    19.67 21
Kennedy 2007 Eric    24.89 12
Kennedy 2007 Bubba   37.88 12
Kennedy 2007 G.L.    25.89 21
Kennedy 2007 Bert    28.89 21
```

```
Kennedy 2008 Clay    26.44 21
Kennedy 2008 Leeann  27.17 17
Kennedy 2008 Georgia 38.90 11
Kennedy 2008 Bill    42.23 25
Kennedy 2008 Holly   18.67 27
Kennedy 2008 Benny   19.09 25
Kennedy 2008 Cammie  28.77 28
Kennedy 2008 Amy     27.08 31
Kennedy 2008 Doris   22.22 24
Kennedy 2008 Robbie  19.80 24
Kennedy 2008 Ted     27.07 25
Kennedy 2008 Sarah   24.44 12
Kennedy 2008 Megan   28.89 11
Kennedy 2008 Jeff    31.11 12
Kennedy 2008 Taz     30.55 11
Kennedy 2008 George  27.56 11
Kennedy 2008 Heather 38.67 15
Kennedy 2009 Nancy   29.90 26
Kennedy 2009 Rusty   30.55 28
Kennedy 2009 Mimi    37.67 22
Kennedy 2009 J.C.    23.33 27
Kennedy 2009 Clark   27.90 25
Kennedy 2009 Rudy    27.78 23
Kennedy 2009 Samuel  34.44 18
Kennedy 2009 Forrest 28.89 26
Kennedy 2009 Luther  72.22 24
Kennedy 2009 Trey     6.78 18
Kennedy 2009 Albert  23.33 19
Kennedy 2009 Che-Min 26.66 33
Kennedy 2009 Preston 32.22 23
Kennedy 2009 Larry   40.00 26
Kennedy 2009 Anton   35.99 28
Kennedy 2009 Sid     27.45 25
Kennedy 2009 Will    28.88 21
Kennedy 2009 Morty   34.44 25
;
```

CUSTOMER_RESPONSE

```
data customer_response;
   input Customer Factor1-Factor4 Source1-Source3
        Quality1-Quality3;
   datalines;
1 . . 1 1 1 1 . 1 . .
2 1 1 . 1 1 1 . 1 1 .
3 . . 1 1 1 1 . . . .
4 1 1 . 1 . 1 . . . 1
5 . 1 . 1 1 . . . . 1
6 . 1 . 1 1 . . . . .
7 . 1 . 1 1 . . 1 . .
8 1 . . 1 1 1 . 1 1 .
9 1 1 . 1 1 . . . . 1
10 1 . . 1 1 1 . 1 1 .
```

```
11 1 1 1 1 . 1 . 1 1 1
12 1 1 . 1 1 1 . . . .
13 1 1 . 1 . 1 . 1 1 .
14 1 1 . 1 1 1 . . . .
15 1 1 . 1 . 1 . 1 1 1
16 1 . . 1 1 . . 1 . .
17 1 1 . 1 1 1 . . 1 .
18 1 1 . 1 1 1 1 . . 1
19 . 1 . 1 1 1 1 . 1 .
20 1 . . 1 1 1 . 1 1 1
21 . . . 1 1 1 . 1 . .
22 . . . 1 1 1 . 1 1 .
23 1 . . 1 . . . . . 1
24 . 1 . 1 1 . . 1 . 1
25 1 1 . 1 1 . . . 1 1
26 1 1 . 1 1 . . 1 . .
27 1 . . 1 1 . . . 1 .
28 1 1 . 1 . . . 1 1 1
29 1 . . 1 1 1 . 1 . 1
30 1 . 1 1 1 . . 1 1 .
31 . . . 1 1 . . 1 1 .
32 1 1 1 1 1 . . 1 1 1
33 1 . . 1 1 . . 1 . 1
34 . . 1 1 . . . 1 1 .
35 1 1 1 1 1 . 1 1 . .
36 1 1 1 1 . 1 . 1 . .
37 1 1 . 1 . . . 1 . .
38 . . . 1 1 1 . 1 . .
39 1 1 . 1 1 . . 1 . 1
40 1 . . 1 . . 1 1 . 1
41 1 . . 1 1 1 1 1 . 1
42 1 1 1 1 . . 1 1 . .
43 1 . . 1 1 1 . 1 . .
44 1 . 1 1 . 1 . 1 . 1
45 . . . 1 . . 1 . . 1
46 . . . 1 1 . . . 1 .
47 1 1 . 1 . . 1 1 . .
48 1 . 1 1 1 . 1 1 . .
49 . . 1 1 1 1 . 1 . 1
50 . 1 . 1 1 . . 1 1 .
51 1 . 1 1 1 1 . . . .
52 1 1 1 1 1 1 . 1 . .
53 . 1 1 1 . 1 . 1 1 1
54 1 . . 1 1 . . 1 1 .
55 1 1 . 1 1 1 . 1 . .
56 1 . . 1 1 . . 1 1 .
57 1 1 . 1 1 . 1 . . 1
58 . 1 . 1 . 1 . . 1 1
59 1 1 1 1 . . 1 1 1 .
60 . 1 1 1 1 1 . . 1 1
61 1 1 1 1 1 1 . 1 . .
62 1 1 . 1 1 . . 1 1 .
63 . . . 1 . . . 1 1 1
64 1 . . 1 1 1 . 1 . .
65 1 . . 1 1 1 . 1 . .
66 1 . . 1 1 1 1 1 1 .
```

```
67 1 1 . 1 1 1 . 1 1 .
68 1 1 . 1 1 1 . 1 1 .
69 1 1 . 1 1 . 1 . . .
70 . . . 1 1 1 . 1 . .
71 1 . . 1 1 . 1 . . 1
72 1 . 1 1 1 1 . . 1 .
73 1 1 . 1 . 1 . 1 1 .
74 1 1 1 1 1 1 . 1 . .
75 . 1 . 1 1 1 . . 1 .
76 1 1 . 1 1 1 . 1 1 1
77 . . . 1 1 1 . . . .
78 1 1 1 1 1 1 . 1 1 .
79 1 . . 1 1 1 . 1 1 .
80 1 1 1 1 1 . 1 1 . 1
81 1 1 . 1 1 1 1 1 1 .
82 . . . 1 1 1 1 . . .
83 1 1 . 1 1 1 . 1 1 .
84 1 . . 1 1 . . 1 1 .
85 . . . 1 . 1 . 1 . .
86 1 . . 1 1 1 . 1 1 1
87 1 1 . 1 1 1 . 1 . .
88 . . . 1 . 1 . . . .
89 1 . . 1 . 1 . . 1 1
90 1 1 . 1 1 1 . 1 . 1
91 . . . 1 1 . . . 1 .
92 1 . . 1 1 1 . 1 1 .
93 1 . . 1 1 . . 1 1 .
94 1 . . 1 1 1 1 1 . .
95 1 . . 1 . 1 1 1 1 .
96 1 . 1 1 1 1 . . 1 .
97 1 1 . 1 1 . . . 1 .
98 1 . 1 1 1 1 1 1 . .
99 1 1 . 1 1 1 1 1 1 .
100 1 . 1 1 1 . . . 1 1
101 1 . 1 1 1 1 . . . .
102 1 . . 1 1 . 1 1 . .
103 1 1 . 1 1 1 . 1 . .
104 . . . 1 1 1 . 1 1 1
105 1 . 1 1 1 . . 1 . 1
106 1 1 1 1 1 1 1 1 1 1
107 1 1 1 1 . . . 1 . 1
108 1 . . 1 . 1 1 1 . .
109 . 1 . 1 1 . . 1 1 .
110 1 . . 1 . . . . . .
111 1 . . 1 1 1 . 1 1 .
112 1 1 . 1 1 1 . . . 1
113 1 1 . 1 1 . 1 1 1 .
114 1 1 . 1 1 . . . . .
115 1 1 . 1 1 . . 1 . .
116 . 1 . 1 1 1 1 1 . .
117 . 1 . 1 1 1 . . . .
118 . 1 1 1 1 . . 1 1 .
119 . . . 1 . . . 1 . .
120 1 1 . 1 . . . . 1 .
;
```

DJIA

```
data djia;
      input Year HighDate date7. High LowDate date7. Low;
      format highdate lowdate date7.;
      datalines;
1968 03DEC68   985.21 21MAR68   825.13
1969 14MAY69   968.85 17DEC69   769.93
1970 29DEC70   842.00 06MAY70   631.16
1971 28APR71   950.82 23NOV71   797.97
1972 11DEC72  1036.27 26JAN72   889.15
1973 11JAN73  1051.70 05DEC73   788.31
1974 13MAR74   891.66 06DEC74   577.60
1975 15JUL75   881.81 02JAN75   632.04
1976 21SEP76  1014.79 02JAN76   858.71
1977 03JAN77   999.75 02NOV77   800.85
1978 08SEP78   907.74 28FEB78   742.12
1979 05OCT79   897.61 07NOV79   796.67
1980 20NOV80  1000.17 21APR80   759.13
1981 27APR81  1024.05 25SEP81   824.01
1982 27DEC82  1070.55 12AUG82   776.92
1983 29NOV83  1287.20 03JAN83  1027.04
1984 06JAN84  1286.64 24JUL84  1086.57
1985 16DEC85  1553.10 04JAN85  1184.96
1986 02DEC86  1955.57 22JAN86  1502.29
1987 25AUG87  2722.42 19OCT87  1738.74
1988 21OCT88  2183.50 20JAN88  1879.14
1989 09OCT89  2791.41 03JAN89  2144.64
1990 16JUL90  2999.75 11OCT90  2365.10
1991 31DEC91  3168.83 09JAN91  2470.30
1992 01JUN92  3413.21 09OCT92  3136.58
1993 29DEC93  3794.33 20JAN93  3241.95
1994 31JAN94  3978.36 04APR94  3593.35
1995 13DEC95  5216.47 30JAN95  3832.08
1996 27DEC96  6560.90 10JAN96  5032.94
1997 06AUG97  8259.30 11APR97  6391.69
1998 23NOV98  9374.27 31AUG98  7539.06
1999 31DEC99 11497.12 22JAN99  9120.67
2000 14JAN00 11722.98 07MAR00  9796.04
2001 21MAY01 11337.92 21SEP01  8235.81
2002 19MAR02 10635.25 09OCT02  7286.27
2003 31DEC03 10453.92 11MAR03  7524.06
2004 28DEC04 10854.54 25OCT04  9749.99
2005 04MAR05 10940.55 20APR05 10012.36
2006 27DEC06 12510.57 20JAN06 10667.39
2007 09OCT07 14164.53 05MAR07 12050.41
2008 02MAY08 13058.20 10OCT08  8451.19
;
```

EDUCATION

```
data education;
   input State $14. +1 Code $ DropoutRate Expenditures MathScore Region $;
   label dropoutrate='Dropout Percentage - 2008'
          expenditures='Expenditure Per Pupil - 2008'
            mathscore='8th Grade Math Exam - 2009';
   datalines;
Alabama        AL 22.3 3197 252 SE
Alaska         AK 35.8 7716 .   W
Arizona        AZ 31.2 3902 259 W
Arkansas       AR 11.5 3273 256 SE
California     CA 32.7 4121 256 W
Colorado       CO 24.7 4408 267 W
Connecticut    CT 16.8 6857 270 NE
Delaware       DE 28.5 5422 261 NE
Florida        FL 38.5 4563 255 SE
Georgia        GA 27.9 3852 258 SE
Hawaii         HI 18.3 4121 251 W
Idaho          ID 21.8 2838 272 W
Illinois       IL 21.5 4906 260 MW
Indiana        IN 13.8 4284 267 MW
Iowa           IA 13.6 4285 278 MW
Kansas         KS 17.9 4443 .   MW
Kentucky       KY 32.7 3347 256 SE
Louisiana      LA 43.1 3317 246 SE
Maine          ME 22.5 4744 .   NE
Maryland       MD 26.0 5758 260 NE
Massachusetts  MA 28.0 5979 .   NE
Michigan       MI 29.3 5116 264 MW
Minnesota      MN 11.4 4755 276 MW
Mississippi    MS 39.9 2874 .   SE
Missouri       MO 26.5 4263 .   MW
Montana        MT 15.0 4293 280 W
Nebraska       NE 13.9 4360 276 MW
Nevada         NV 28.1 3791 .   W
New Hampshire  NH 25.9 4807 273 NE
New Jersey     NE 20.4 7549 269 NE
New Mexico     NM 28.5 3473 256 W
New York       NY 35.0 .    261 NE
North Carolina NC 31.2 3874 250 SE
North Dakota   ND 12.1 3952 281 MW
Ohio           OH 24.4 4649 264 MW
;
```

EMPDATA

```
data empdata;
input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
```

```
      City $ 30-42 State $ 43-44 /
      Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date7.
      @43 Hired date7. HomePhone $ 54-65;
format birth hired date7.;
datalines;
1919    Adams      Gerald    Stamford     CT
M       TA2        34376     15SEP70      07JUN05    203-781-1255
1653    Alexander  Susan     Bridgeport   CT
F       ME2        35108     18OCT72      12AUG98    203-675-7715
1400    Apple      Troy      New York     NY
M       ME1        29769     08NOV85      19OCT06    212-586-0808
1350    Arthur     Barbara   New York     NY
F       FA3        32886     03SEP63      01AUG00    718-383-1549
1401    Avery      Jerry     Paterson     NJ
M       TA3        38822     16DEC68      20NOV93    201-732-8787
1499    Barefoot   Joseph    Princeton    NJ
M       ME3        43025     29APR62      10JUN95    201-812-5665
1101    Baucom     Walter    New York     NY
M       SCP        18723     09JUN80      04OCT98    212-586-8060
1333    Blair      Justin    Stamford     CT
M       PT2        88606     02APR79      13FEB03    203-781-1777
1402    Blalock    Ralph     New York     NY
M       TA2        32615     20JAN71      05DEC98    718-384-2849
1479    Bostic     Marie     New York     NY
F       TA3        38785     25DEC66      08OCT03    718-384-8816
1403    Bowden     Earl      Bridgeport   CT
M       ME1        28072     31JAN79      24DEC99    203-675-3434
1739    Boyce      Jonathan  New York     NY
M       PT1        66517     28DEC82      30JAN00    212-587-1247
1658    Bradley    Jeremy    New York     NY
M       SCP        17943     11APR65      03MAR00    212-587-3622
1428    Brady      Christine Stamford     CT
F       PT1        68767     07APR80      19NOV02    203-781-1212
1782    Brown      Jason     Stamford     CT
M       ME2        35345     07DEC73      25FEB00    203-781-0019
1244    Bryant     Leonard   New York     NY
M       ME2        36925     03SEP71      20JAN96    718-383-3334
1383    Burnette   Thomas    New York     NY
M       BCK        25823     28JAN76      23OCT00    718-384-3569
1574    Cahill     Marshall  New York     NY
M       FA2        28572     30APR74      23DEC97    718-383-2338
1789    Caraway    Davis     New York     NY
M       SCP        18326     28JAN85      14APR04    212-587-9000
1404    Carter     Donald    New York     NY
M       PT2        91376     27FEB71      04JAN98    718-384-2946
1437    Carter     Dorothy   Bridgeport   CT
F       A3         33104     23SEP68      03SEP92    203-675-4117
1639    Carter     Karen     Stamford     CT
F       A3         40260     29JUN65      31JAN92    203-781-8839
1269    Caston     Franklin  Stamford     CT
M       NA1        41690     06MAY80      01DEC00    203-781-3335
1065    Chapman    Neil      New York     NY
M       ME2        35090     29JAN72      10JAN95    718-384-5618
1876    Chin       Jack      New York     NY
M       TA3        39675     23MAY66      30APR96    212-588-5634
1037    Chow       Jane      Stamford     CT
```

```
F       TA1        28558     13APR82      16SEP04    203-781-8868
1129    Cook       Brenda    New York     NY
F       ME2        34929     11DEC79      20AUG03    718-383-2313
1988    Cooper     Anthony   New York     NY
M       FA3        32217     03DEC57      21SEP92    212-587-1228
1405    Davidson   Jason     Paterson     NJ
M       SCP        18056     08MAR54      29JAN00    201-732-2323
1430    Dean       Sandra    Bridgeport   CT
F       TA2        32925     03MAR70      30APR05    203-675-1647
1983    Dean       Sharon    New York     NY
F       FA3        33419     03MAR50      30APR85    718-384-1647
1134    Delgado    Maria     Stamford     CT
F       TA2        33462     08MAR77      24DEC04    203-781-1528
1118    Dennis     Roger     New York     NY
M       PT3        111379    19JAN57      21DEC88    718-383-1122
1438    Donaldson  Karen     Stamford     CT
F       TA3        39223     18MAR63      21NOV03    203-781-2229
1125    Dunlap     Donna     New York     NY
F       FA2        28888     11NOV76      14DEC95    718-383-2094
1475    Eaton      Alicia    New York     NY
F       FA2        27787     18DEC71      16JUL98    718-383-2828
1117    Edgerton   Joshua    New York     NY
M       TA3        39771     08JUN56      16AUG00    212-588-1239
1935    Fernandez  Katrina   Bridgeport   CT
F       NA2        51081     31MAR72      19OCT01    203-675-2962
1124    Fields     Diana     White Plains NY
F       FA1        23177     13JUL82      04OCT01    914-455-2998
1422    Fletcher   Marie     Princeton    NJ
F       FA1        22454     07JUN79      09APR99    201-812-0902
1616    Flowers    Annette   New York     NY
F       TA2        34137     04MAR68      07JUN01    718-384-3329
1406    Foster     Gerald    Bridgeport   CT
M       ME2        35185     11MAR69      20FEB95    203-675-6363
1120    Garcia     Jack      New York     NY
M       ME1        28619     14SEP80      10OCT01    718-384-4930
1094    Gomez      Alan      Bridgeport   CT
M       FA1        22268     05APR78      20APR99    203-675-7181
1389    Gordon     Levi      New York     NY
M       BCK        25028     18JUL67      21AUG03    718-384-9326
1905    Graham     Alvin     New York     NY
M       PT1        65111     19APR80      01JUN00    212-586-8815
1407    Grant      Daniel    Mt. Vernon   NY
M       PT1        68096     26MAR77      21MAR98    914-468-1616
1114    Green      Janice    New York     NY
F       TA2        32928     21SEP77      30JUN06    212-588-1092
;
run;
```

```
data empdata;
input IdNumber $ 1-4 LastName $ 8-18 FirstName $ 19-28
      City $ 29-41 State $ 42-43 /
      Gender $ 45 JobCode $ 49-51 Salary 55-60 @63 Birth date7.
      @73 Hired date7. HomePhone $ 83-95;
format birth hired date7.;
datalines;
1919   Adams      Gerald    Stamford     CT M   TA2   34376   15SEP70   07JUN05   203/781-1255
```

```
1653   Alexander  Susan     Bridgeport   CT F   ME2   35108   18OCT72   12AUG98   203/675-7715
1400   Apple      Troy      New York     NY M   ME1   29769   08NOV85   19OCT06   212/586-0808
1350   Arthur     Barbara   New York     NY F   FA3   32886   03SEP63   01AUG00   718/383-1549
1401   Avery      Jerry     Paterson     NJ M   TA3   38822   16DEC68   20NOV93   201/732-8787
1499   Barefoot   Joseph    Princeton    NJ M   ME3   43025   29APR62   10JUN95   201/812-5665
1101   Baucom     Walter    New York     NY M   SCP   18723   09JUN80   04OCT98   212/586-8060
1333   Blair      Justin    Stamford     CT M   PT2   88606   02APR79   13FEB03   203/781-1777
1402   Blalock    Ralph     New York     NY M   TA2   32615   20JAN71   05DEC98   718/384-2849
1479   Bostic     Marie     New York     NY F   TA3   38785   25DEC66   08OCT03   718/384-8816
1403   Bowden     Earl      Bridgeport   CT M   ME1   28072   31JAN79   24DEC99   203/675-3434
1739   Boyce      Jonathan  New York     NY M   PT1   66517   28DEC82   30JAN00   212/587-1247
1658   Bradley    Jeremy    New York     NY M   SCP   17943   11APR65   03MAR00   212/587-3622
1428   Brady      Christine Stamford     CT F   PT1   68767   07APR80   19NOV02   203/781-1212
1782   Brown      Jason     Stamford     CT M   ME2   35345   07DEC73   25FEB00   203/781-0019
1244   Bryant     Leonard   New York     NY M   ME2   36925   03SEP71   20JAN96   718/383-3334
1383   Burnette   Thomas    New York     NY M   BCK   25823   28JAN76   23OCT00   718/384-3569
1574   Cahill     Marshall  New York     NY M   FA2   28572   30APR74   23DEC97   718/383-2338
1789   Caraway    Davis     New York     NY M   SCP   18326   28JAN85   14APR04   212/587-9000
1404   Carter     Donald    New York     NY M   PT2   91376   27FEB71   04JAN98   718/384-2946
1437   Carter     Dorothy   Bridgeport   CT F   A3    33104   23SEP68   03SEP92   203/675-4117
1639   Carter     Karen     Stamford     CT F   A3    40260   29JUN65   31JAN92   203/781-8839
1269   Caston     Franklin  Stamford     CT M   NA1   41690   06MAY80   01DEC00   203/781-3335
1065   Chapman    Neil      New York     NY M   ME2   35090   29JAN72   10JAN95   718/384-5618
1876   Chin       Jack      New York     NY M   TA3   39675   23MAY66   30APR96   212/588-5634
1037   Chow       Jane      Stamford     CT F   TA1   28558   13APR82   16SEP04   203/781-8868
1129   Cook       Brenda    New York     NY F   ME2   34929   11DEC79   20AUG03   718/383-2313
1988   Cooper     Anthony   New York     NY M   FA3   32217   03DEC57   21SEP92   212/587-1228
1405   Davidson   Jason     Paterson     NJ M   SCP   18056   08MAR54   29JAN00   201/732-2323
1430   Dean       Sandra    Bridgeport   CT F   TA2   32925   03MAR70   30APR05   203/675-1647
1983   Dean       Sharon    New York     NY F   FA3   33419   03MAR50   30APR85   718/384-1647
1134   Delgado    Maria     Stamford     CT F   TA2   33462   08MAR77   24DEC04   203/781-1528
1118   Dennis     Roger     New York     NY M   PT3   111379  19JAN57   21DEC88   718/383-1122
1438   Donaldson  Karen     Stamford     CT F   TA3   39223   18MAR63   21NOV03   203/781-2229
1125   Dunlap     Donna     New York     NY F   FA2   28888   11NOV76   14DEC95   718/383-2094
1475   Eaton      Alicia    New York     NY F   FA2   27787   18DEC71   16JUL98   718/383-2828
1117   Edgerton   Joshua    New York     NY M   TA3   39771   08JUN56   16AUG00   212/588-1239
1935   Fernandez  Katrina   Bridgeport   CT F   NA2   51081   31MAR72   19OCT01   203/675-2962
1124   Fields     Diana     White Plains NY F   FA1   23177   13JUL82   04OCT01   914/455-2998
1422   Fletcher   Marie     Princeton    NJ F   FA1   22454   07JUN79   09APR99   201/812-0902
1616   Flowers    Annette   New York     NY F   TA2   34137   04MAR68   07JUN01   718/384-3329
1406   Foster     Gerald    Bridgeport   CT M   ME2   35185   11MAR69   20FEB95   203/675-6363
1120   Garcia     Jack      New York     NY M   ME1   28619   14SEP80   10OCT01   718/384-4930
1094   Gomez      Alan      Bridgeport   CT M   FA1   22268   05APR78   20APR99   203/675-7181
1389   Gordon     Levi      New York     NY M   BCK   25028   18JUL67   21AUG03   718/384-9326
1905   Graham     Alvin     New York     NY M   PT1   65111   19APR80   01JUN00   212/586-8815
1407   Grant      Daniel    Mt. Vernon   NY M   PT1   68096   26MAR77   21MAR98   914/468-1616
1114   Green      Janice    New York     NY F   TA2   32928   21SEP77   30JUN06   212/588-1092
;
```

ENERGY

```
data energy;
   length State $2;
```

```
   input Region Division state $ Type Expenditures;
   datalines;
1 1 ME 1 708
1 1 ME 2 379
1 1 NH 1 597
1 1 NH 2 301
1 1 VT 1 353
1 1 VT 2 188
1 1 MA 1 3264
1 1 MA 2 2498
1 1 RI 1 531
1 1 RI 2 358
1 1 CT 1 2024
1 1 CT 2 1405
1 2 NY 1 8786
1 2 NY 2 7825
1 2 NJ 1 4115
1 2 NJ 2 3558
1 2 PA 1 6478
1 2 PA 2 3695
4 3 MT 1 322
4 3 MT 2 232
4 3 ID 1 392
4 3 ID 2 298
4 3 WY 1 194
4 3 WY 2 184
4 3 CO 1 1215
4 3 CO 2 1173
4 3 NM 1 545
4 3 NM 2 578
4 3 AZ 1 1694
4 3 AZ 2 1448
4 3 UT 1 621
4 3 UT 2 438
4 3 NV 1 493
4 3 NV 2 378
4 4 WA 1 1680
4 4 WA 2 1122
4 4 OR 1 1014
4 4 OR 2 756
4 4 CA 1 10643
4 4 CA 2 10114
4 4 AK 1 349
4 4 AK 2 329
4 4 HI 1 273
4 4 HI 2 298
;
```

EXP Library

EXP.RESULTS

The following sections are the raw data and DATA steps for the EXP library.

```
options ps=40 ls=64 nodate pageno=1;

LIBNAME exp 'library-name';

data exp.results;
   set exp.wght(firstobs=1 obs=11 keep=id treat initwt wt3mos
       age);
   if age>100 then delete;
run;
proc print data=exp.results noobs;
    title 'The RESULTS Data Set';
run;
proc datasets library=exp;

data exp.results;
    input id  treat $  initwt  wt3mos age;
    datalines;
 1    Other      166.28    146.98     35
 2    Other      214.42    210.22     54
 3    Other      172.46    159.42     33
 5    Other      175.41    160.66     37
 6    Other      173.13    169.40     20
 7    Other      181.25    170.94     30
10    Other      239.83    214.48     48
11    Other      175.32    162.66     51
12    Other      227.01    211.06     29
13    Other      274.82    251.82     31
;
run;
```

EXP.SUR

```
data exp.sur;
    input id  treat $  initwt  wt3mos  wt6mos  age;
    datalines;
14    surgery    203.60    169.78    143.88     38
17    surgery    171.52    150.33    123.18     42
18    surgery    207.46    155.22       .       41
;
run;
```

EXPREV

```
ods html close;
data exprev;
input Country $ 1-24 Emp_ID $ 25-32 Order_Date $  Ship_Date $  Sale_Type $ 67-75 Quantity Price Cost;

datalines;
Antarctica              99999999     1/1/16      1/7/16            Internet    2     92.60      20.70
Puerto Rico             99999999     1/1/16      1/5/16            Catalog    14    51.20      12.10
Virgin Islands (U.S.)   99999999     1/1/16      1/4/16            In Store   25    31.10      15.65
Aruba                   99999999     1/1/16      1/4/16            Catalog    30    123.70     59.00
Bahamas                 99999999     1/1/16      1/4/16            Catalog     8    113.40     28.45
Bermuda                 99999999     1/1/16      1/4/16            Catalog     7    41.00      9.25
Belize                  120458       1/2/16      1/2/16            In Store    2    146.40     36.70
British Virgin Islands  99999999     1/2/16      1/5/16            Catalog    11    40.20      20.20
Canada                  99999999     1/2/16      1/5/16            Catalog    100   11.80      5.00
Cayman Islands          120454       1/2/16      1/2/16            In Store   20    71.00      32.30
Costa Rica              99999999     1/2/16      1/6/16            Internet   31    53.00      26.60
Cuba                    121044       1/2/16      1/2/16            Internet   12    42.40      19.35
Dominican Republic      121040       1/2/16      1/2/16            Internet   13    48.00      23.95
El Salvador             99999999     1/2/16      1/6/16            Catalog    21    266.40     66.70
Guatemala               120931       1/2/16      1/2/16            In Store   13    144.40     65.70
Haiti                   121059       1/2/16      1/2/16            Internet    5    47.90      23.45
Honduras                120455       1/2/16      1/2/16            Internet   20    66.40      30.25
Jamaica                 99999999     1/2/16      1/4/16            In Store   23    169.80     38.70
Mexico                  120127       1/2/16      1/2/16            In Store   30    211.80     33.65
Montserrat              120127       1/2/16      1/2/16            In Store   19    184.20     36.90
Nicaragua               120932       1/2/16      1/2/16            Internet   16    122.00     28.75
Panama                  99999999     1/2/16      1/6/16            Internet   20    88.20      38.40
Saint Kitts/Nevis       99999999     1/2/16      1/6/16            Internet   20    41.40      18.00
St. Helena              120360       1/2/16      1/2/16            Internet   19    94.70      47.45
St. Pierre/Miquelon     120842       1/2/16      1/16/16           Internet   16    103.80     47.25
Turks/Caicos Islands    120372       1/2/16      1/2/16            Internet   10    57.70      28.95
United States           120372       1/2/16      1/2/16            Internet   20    88.20      38.40
Anguilla                99999999     1/2/16      1/6/16            In Store   15    233.50     22.25
Antigua/Barbuda         120458       1/2/16      1/2/16            In Store   31    99.60      45.35
Argentina               99999999     1/2/16      1/6/16            In Store   42    408.80     87.15
Barbados                99999999     1/2/16      1/6/16            In Store   26    94.80      42.60
Bolivia                 120127       1/2/16      1/2/16            In Store   26    66.00      16.60
Brazil                  120127       1/2/16      1/2/16            Catalog    12    73.40      18.45
Chile                   120447       1/2/16      1/2/16            In Store   20    19.10      8.75
Colombia                121059       1/2/16      1/2/16            Internet   28    361.40     90.45
Dominica                121043       1/2/16      1/2/16            Internet   35    121.30     57.80
Ecuador                 121042       1/2/16      1/2/16            In Store   11    100.90     50.55
Falkland Islands        120932       1/2/16      1/2/16            In Store   15    61.40      30.80
French Guiana           120935       1/2/16      1/2/16            Catalog    15    96.40      43.85
Grenada                 120931       1/2/16      1/2/16            Catalog    19    56.30      25.05
Guadeloupe              120445       1/2/16      1/2/16            Internet   21    231.60     48.70
Guyana                  120455       1/2/16      1/2/16            In Store   25    132.80     30.25
Martinique              120841       1/2/16      1/3/16            In Store   16    56.30      31.05
Netherlands Antilles    99999999     1/2/16      1/6/16            In Store   31    41.80      19.45
Paraguay                120603       1/2/16      1/2/16            Catalog    17    117.60     58.90
```

```
Peru                    120845       1/2/16      1/2/16            Catalog    12    93.80      41.75
St. Lucia               120845       1/2/16      1/2/16            Internet   19    64.30      28.65
Suriname                120538       1/3/16      1/3/16            Internet   22    110.80     29.35
;
run;
```

GROC

```
data groc;
   input Region $9. Manager $ Department $ Sales;
   datalines;
Southeast    Hayes       Paper       250
Southeast    Hayes       Produce     100
Southeast    Hayes       Canned      120
Southeast    Hayes       Meat         80
Southeast    Michaels    Paper        40
Southeast    Michaels    Produce     300
Southeast    Michaels    Canned      220
Southeast    Michaels    Meat         70
Northwest    Jeffreys    Paper        60
Northwest    Jeffreys    Produce     600
Northwest    Jeffreys    Canned      420
Northwest    Jeffreys    Meat         30
Northwest    Duncan      Paper        45
Northwest    Duncan      Produce     250
Northwest    Duncan      Canned      230
Northwest    Duncan      Meat         73
Northwest    Aikmann     Paper        45
Northwest    Aikmann     Produce     205
Northwest    Aikmann     Canned      420
Northwest    Aikmann     Meat         76
Southwest    Royster     Paper        53
Southwest    Royster     Produce     130
Southwest    Royster     Canned      120
Southwest    Royster     Meat         50
Southwest    Patel       Paper        40
Southwest    Patel       Produce     350
Southwest    Patel       Canned      225
Southwest    Patel       Meat         80
Northeast    Rice        Paper        90
Northeast    Rice        Produce      90
Northeast    Rice        Canned      420
Northeast    Rice        Meat         86
Northeast    Fuller      Paper       200
Northeast    Fuller      Produce     300
Northeast    Fuller      Canned      420
Northeast    Fuller      Meat        125
;
```

MATCH_11

```
data match_11;
   input Pair Low Age Lwt Race Smoke Ptd Ht UI @@;
   select(race);
      when (1) do;
         race1=0;
         race2=0;
      end;
      when (2) do;
         race1=1;
         race2=0;
      end;
      when (3) do;
         race1=0;
         race2=1;
      end;
   end;
   datalines;
1  0 14 135 1 0 0 0 0     1  1 14 101 3 1 1 0 0
2  0 15  98 2 0 0 0 0     2  1 15 115 3 0 0 0 1
3  0 16  95 3 0 0 0 0     3  1 16 130 3 0 0 0 0
4  0 17 103 3 0 0 0 0     4  1 17 130 3 1 1 0 1
5  0 17 122 1 1 0 0 0     5  1 17 110 1 1 0 0 0
6  0 17 113 2 0 0 0 0     6  1 17 120 1 1 0 0 0
7  0 17 113 2 0 0 0 0     7  1 17 120 2 0 0 0 0
8  0 17 119 3 0 0 0 0     8  1 17 142 2 0 0 1 0
9  0 18 100 1 1 0 0 0     9  1 18 148 3 0 0 0 0
10 0 18  90 1 1 0 0 1     10 1 18 110 2 1 1 0 0
11 0 19 150 3 0 0 0 0     11 1 19  91 1 1 1 0 1
12 0 19 115 3 0 0 0 0     12 1 19 102 1 0 0 0 0
13 0 19 235 1 1 0 1 0     13 1 19 112 1 1 0 0 1
14 0 20 120 3 0 0 0 1     14 1 20 150 1 1 0 0 0
15 0 20 103 3 0 0 0 0     15 1 20 125 3 0 0 0 1
16 0 20 169 3 0 1 0 1     16 1 20 120 2 1 0 0 0
17 0 20 141 1 0 1 0 1     17 1 20  80 3 1 0 0 1
18 0 20 121 2 1 0 0 0     18 1 20 109 3 0 0 0 0
19 0 20 127 3 0 0 0 0     19 1 20 121 1 1 1 0 1
20 0 20 120 3 0 0 0 0     20 1 20 122 2 1 0 0 0
21 0 20 158 1 0 0 0 0     21 1 20 105 3 0 0 0 0
22 0 21 108 1 1 0 0 1     22 1 21 165 1 1 0 1 0
23 0 21 124 3 0 0 0 0     23 1 21 200 2 0 0 0 0
24 0 21 185 2 1 0 0 0     24 1 21 103 3 0 0 0 0
25 0 21 160 1 0 0 0 0     25 1 21 100 3 0 1 0 0
26 0 21 115 1 0 0 0 0     26 1 21 130 1 1 0 1 0
27 0 22  95 3 0 0 1 0     27 1 22 130 1 1 0 0 0
28 0 22 158 2 0 1 0 0     28 1 22 130 1 1 1 0 1
29 0 23 130 2 0 0 0 0     29 1 23  97 3 0 0 0 1
30 0 23 128 3 0 0 0 0     30 1 23 187 2 1 0 0 0
31 0 23 119 3 0 0 0 0     31 1 23 120 3 0 0 0 0
32 0 23 115 3 1 0 0 0     32 1 23 110 1 1 1 0 0
33 0 23 190 1 0 0 0 0     33 1 23  94 3 1 0 0 0
```

```
34 0 24  90 1 1 1 0 0     34 1 24 128 2 0 1 0 0
35 0 24 115 1 0 0 0 0     35 1 24 132 3 0 0 1 0
36 0 24 110 3 0 0 0 0     36 1 24 155 1 1 1 0 0
37 0 24 115 3 0 0 0 0     37 1 24 138 1 0 0 0 0
38 0 24 110 3 0 1 0 0     38 1 24 105 2 1 0 0 0
39 0 25 118 1 1 0 0 0     39 1 25 105 3 0 1 1 0
40 0 25 120 3 0 0 0 1     40 1 25  85 3 0 0 0 1
41 0 25 155 1 0 0 0 0     41 1 25 115 3 0 0 0 0
42 0 25 125 2 0 0 0 0     42 1 25  92 1 1 0 0 0
43 0 25 140 1 0 0 0 0     43 1 25  89 3 0 1 0 0
44 0 25 241 2 0 0 1 0     44 1 25 105 3 0 1 0 0
45 0 26 113 1 1 0 0 0     45 1 26 117 1 1 1 0 0
46 0 26 168 2 1 0 0 0     46 1 26  96 3 0 0 0 0
47 0 26 133 3 1 1 0 0     47 1 26 154 3 0 1 1 0
48 0 26 160 3 0 0 0 0     48 1 26 190 1 1 0 0 0
49 0 27 124 1 1 0 0 0     49 1 27 130 2 0 0 0 1
50 0 28 120 3 0 0 0 0     50 1 28 120 3 1 1 0 1
51 0 28 130 3 0 0 0 0     51 1 28  95 1 1 0 0 0
52 0 29 135 1 0 0 0 0     52 1 29 130 1 0 0 0 1
53 0 30  95 1 1 0 0 0     53 1 30 142 1 1 1 0 0
54 0 31 215 1 1 0 0 0     54 1 31 102 1 1 1 0 0
55 0 32 121 3 0 0 0 0     55 1 32 105 1 1 0 0 0
56 0 34 170 1 0 1 0 0     56 1 34 187 2 1 0 1 0
;
```

PROCLIB.DELAY

```
data proclib.delay;
   input flight $3. +5 date date7. +2 orig $3. +3 dest $3. +3
         delaycat $15. +2 destype $15. +8 delay;
   informat date date7.;
   format date date7.;
   datalines;
114     01MAR08  LGA   LAX   1-10 Minutes     Domestic               8
202     01MAR08  LGA   ORD   No Delay         Domestic              -5
219     01MAR08  LGA   LON   11+ Minutes      International          18
622     01MAR08  LGA   FRA   No Delay         International         -5
132     01MAR08  LGA   YYZ   11+ Minutes      International          14
271     01MAR08  LGA   PAR   1-10 Minutes     International           5
302     01MAR08  LGA   WAS   No Delay         Domestic              -2
114     02MAR08  LGA   LAX   No Delay         Domestic               0
202     02MAR08  LGA   ORD   1-10 Minutes     Domestic               5
219     02MAR08  LGA   LON   11+ Minutes      International          18
622     02MAR08  LGA   FRA   No Delay         International           0
132     02MAR08  LGA   YYZ   1-10 Minutes     International           5
271     02MAR08  LGA   PAR   1-10 Minutes     International           4
302     02MAR08  LGA   WAS   No Delay         Domestic               0
114     03MAR08  LGA   LAX   No Delay         Domestic              -1
202     03MAR08  LGA   ORD   No Delay         Domestic              -1
219     03MAR08  LGA   LON   1-10 Minutes     International           4
622     03MAR08  LGA   FRA   No Delay         International         -2
132     03MAR08  LGA   YYZ   1-10 Minutes     International           6
271     03MAR08  LGA   PAR   1-10 Minutes     International           2
```

```
302     03MAR08  LGA   WAS   1-10 Minutes     Domestic               5
114     04MAR08  LGA   LAX   11+ Minutes      Domestic              15
202     04MAR08  LGA   ORD   No Delay         Domestic              -5
219     04MAR08  LGA   LON   1-10 Minutes     International           3
622     04MAR08  LGA   FRA   11+ Minutes      International          30
132     04MAR08  LGA   YYZ   No Delay         International          -5
271     04MAR08  LGA   PAR   1-10 Minutes     International           5
302     04MAR08  LGA   WAS   1-10 Minutes     Domestic               7
114     05MAR08  LGA   LAX   No Delay         Domestic              -2
202     05MAR08  LGA   ORD   1-10 Minutes     Domestic               2
219     05MAR08  LGA   LON   1-10 Minutes     International           3
622     05MAR08  LGA   FRA   No Delay         International          -6
132     05MAR08  LGA   YYZ   1-10 Minutes     International           3
271     05MAR08  LGA   PAR   1-10 Minutes     International           5
114     06MAR08  LGA   LAX   No Delay         Domestic              -1
202     06MAR08  LGA   ORD   No Delay         Domestic              -3
219     06MAR08  LGA   LON   11+ Minutes      International          27
132     06MAR08  LGA   YYZ   1-10 Minutes     International           7
302     06MAR08  LGA   WAS   1-10 Minutes     Domestic               1
114     07MAR08  LGA   LAX   No Delay         Domestic              -1
202     07MAR08  LGA   ORD   No Delay         Domestic              -2
219     07MAR08  LGA   LON   11+ Minutes      International          15
622     07MAR08  LGA   FRA   11+ Minutes      International          21
132     07MAR08  LGA   YYZ   No Delay         International          -2
271     07MAR08  LGA   PAR   1-10 Minutes     International           4
302     07MAR08  LGA   WAS   No Delay         Domestic               0
;
```

PROCLIB.EMP95

```
data proclib.emp95;
   input #1 idnum $4. @6 name $15.
         #2 address $42.
         #3 salary 6.;
   datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane  Garner NC 27509
33190
2776 Robert Jones
12988 Wellington Farms Ave. Cary NC 27512
29025
8699 Jerry Capalleti
222 West L St. Oxford NC 27587
39985
2100 Lanny Engles
293 Manning Pl. Raleigh NC 27606
30998
9857 Kathy Krupski
1000 Taft Ave. Morrisville NC 27508
38756
```

```
0987 Dolly Lunford
2344 Persimmons Branch  Apex NC 27505
44010
3286 Hoa Nguyen
2818 Long St. Cary NC 27513
87734
6579 Bryan Samosky
3887 Charles Ave. Garner NC 27508
50234
3888 Kim Siu
5662 Magnolia Blvd Southeast Cary NC 27513
77558
;
```

PROCLIB.EMP96

```
data proclib.emp96;
   input #1 idnum $4. @6 name $15.
         #2 address $42.
         #3 salary 6.;
   datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane  Garner NC 27509
33190
2776 Robert Jones
12988 Wellington Farms Ave. Cary NC 27511
29025
8699 Jerry Capalleti
222 West L St. Oxford NC 27587
39985
3278 Mary Cravens
211 N. Cypress St. Cary NC 27512
35362
2100 Lanny Engles
293 Manning Pl. Raleigh NC 27606
30998
9857 Kathy Krupski
100 Taft Ave. Morrisville NC 27508
40456
0987 Dolly Lunford
2344 Persimmons Branch Trail Apex NC 27505
45110
3286 Hoa Nguyen
2818 Long St. Cary NC 27513
89834
6579 Bryan Samosky
3887 Charles Ave. Garner NC 27508
50234
3888 Kim Siu
5662 Magnolia Blvd Southwest Cary NC 27513
```

```
79958
6544 Roger Monday
3004 Crepe Myrtle Court Raleigh NC 27604
47007
;
```

PROCLIB.INTERNAT

```
data proclib.internat;
   input flight $3.  +5 date date7. +2 dest $3. +8 boarded;
   informat date date7.;
   format date date7.;
   datalines;
219     01MAR08  LON        198
622     01MAR08  FRA        207
132     01MAR08  YYZ        115
271     01MAR08  PAR        138
219     02MAR08  LON        147
622     02MAR08  FRA        176
132     02MAR08  YYZ        106
271     02MAR08  PAR        172
219     03MAR08  LON        197
622     03MAR08  FRA        180
132     03MAR08  YYZ         75
271     03MAR08  PAR        147
219     04MAR08  LON        232
622     04MAR08  FRA        137
132     04MAR08  YYZ        117
271     04MAR08  PAR        146
219     05MAR08  LON        160
622     05MAR08  FRA        185
132     05MAR08  YYZ        157
271     05MAR08  PAR        177
219     06MAR08  LON        163
132     06MAR08  YYZ        150
219     07MAR08  LON        241
622     07MAR08  FRA        210
132     07MAR08  YYZ        164
271     07MAR08  PAR        155
;
```

PROCLIB.LAKES

```
data proclib.lakes;
    input region $ 1-2 lake $ 5-13 pol_a1 pol_a2 pol_b1-pol_b4;
    datalines;
NE  Carr       0.24     0.99     0.95     0.36     0.44     0.67
NE  Duraleigh  0.34     0.01     0.48     0.58     0.12     0.56
NE  Charlie    0.40     0.48     0.29     0.56     0.52     0.95
NE  Farmer     0.60     0.65     0.25     0.20     0.30     0.64
```

```
NW  Canyon     0.63     0.44     0.20     0.98     0.19     0.01
NW  Morris     0.85     0.95     0.80     0.67     0.32     0.81
NW  Golf       0.69     0.37     0.08     0.72     0.71     0.32
NW  Falls      0.01     0.02     0.59     0.58     0.67     0.02
SE  Pleasant   0.16     0.96     0.71     0.35     0.35     0.48
SE  Juliette   0.82     0.35     0.09     0.03     0.59     0.90
SE  Massey     1.01     0.77     0.45     0.32     0.55     0.66
SE  Delta      0.84     1.05     0.90     0.09     0.64     0.03
SW  Alumni     0.45     0.32     0.45     0.44     0.55     0.12
SW  New Dam    0.80     0.70     0.31     0.98     1.00     0.22
SW  Border     0.51     0.04     0.55     0.35     0.45     0.78
SW  Red        0.22     0.09     0.02     0.10     0.32     0.01
;
```

PROCLIB.PAYROLL

This data set (table) is updated in "Updating Data in a PROC SQL Table" in *SAS Viya SQL Procedure User's Guide*.

```
data proclib.payroll;
   input IdNumber $4. +3 Gender $1. +4 Jobcode $3. +9 Salary 5.
         +2 Birth date7. +2 Hired date7.;
   informat birth date7. hired date7.;
   format birth date7. hired date7.;
   datalines;
1919   M    TA2         34376  12SEP60  04JUN87
1653   F    ME2         35108  15OCT64  09AUG90
1400   M    ME1         29769  05NOV67  16OCT90
1350   F    FA3         32886  31AUG65  29JUL90
1401   M    TA3         38822  13DEC50  17NOV85
1499   M    ME3         43025  26APR54  07JUN80
1101   M    SCP         18723  06JUN62  01OCT90
1333   M    PT2         88606  30MAR61  10FEB81
1402   M    TA2         32615  17JAN63  02DEC90
1479   F    TA3         38785  22DEC68  05OCT89
1403   M    ME1         28072  28JAN69  21DEC91
1739   M    PT1         66517  25DEC64  27JAN91
1658   M    SCP         17943  08APR67  29FEB92
1428   F    PT1         68767  04APR60  16NOV91
1782   M    ME2         35345  04DEC70  22FEB92
1244   M    ME2         36925  31AUG63  17JAN88
1383   M    BCK         25823  25JAN68  20OCT92
1574   M    FA2         28572  27APR60  20DEC92
1789   M    SCP         18326  25JAN57  11APR78
1404   M    PT2         91376  24FEB53  01JAN80
1437   F    FA3         33104  20SEP60  31AUG84
1639   F    TA3         40260  26JUN57  28JAN84
1269   M    NA1         41690  03MAY72  28NOV92
1065   M    ME2         35090  26JAN44  07JAN87
1876   M    TA3         39675  20MAY58  27APR85
1037   F    TA1         28558  10APR64  13SEP92
1129   F    ME2         34929  08DEC61  17AUG91
1988   M    FA3         32217  30NOV59  18SEP84
```

1405	M	SCP	18056	05MAR66	26JAN92
1430	F	TA2	32925	28FEB62	27APR87
1983	F	FA3	33419	28FEB62	27APR87
1134	F	TA2	33462	05MAR69	21DEC88
1118	M	PT3	111379	16JAN44	18DEC80
1438	F	TA3	39223	15MAR65	18NOV87
1125	F	FA2	28888	08NOV68	11DEC87
1475	F	FA2	27787	15DEC61	13JUL90
1117	M	TA3	39771	05JUN63	13AUG92
1935	F	NA2	51081	28MAR54	16OCT81
1124	F	FA1	23177	10JUL58	01OCT90
1422	F	FA1	22454	04JUN64	06APR91
1616	F	TA2	34137	01MAR70	04JUN93
1406	M	ME2	35185	08MAR61	17FEB87
1120	M	ME1	28619	11SEP72	07OCT93
1094	M	FA1	22268	02APR70	17APR91
1389	M	BCK	25028	15JUL59	18AUG90
1905	M	PT1	65111	16APR72	29MAY92
1407	M	PT1	68096	23MAR69	18MAR90
1114	F	TA2	32928	18SEP69	27JUN87
1410	M	PT2	84685	03MAY67	07NOV86
1439	F	PT1	70736	06MAR64	10SEP90
1409	M	ME3	41551	19APR50	22OCT81
1408	M	TA2	34138	29MAR60	14OCT87
1121	M	ME1	29112	26SEP71	07DEC91
1991	F	TA1	27645	07MAY72	12DEC92
1102	M	TA2	34542	01OCT59	15APR91
1356	M	ME2	36869	26SEP57	22FEB83
1545	M	PT1	66130	12AUG59	29MAY90
1292	F	ME2	36691	28OCT64	02JUL89
1440	F	ME2	35757	27SEP62	09APR91
1368	M	FA2	27808	11JUN61	03NOV84
1369	M	TA2	33705	28DEC61	13MAR87
1411	M	FA2	27265	27MAY61	01DEC89
1113	F	FA1	22367	15JAN68	17OCT91
1704	M	BCK	25465	30AUG66	28JUN87
1900	M	ME2	35105	25MAY62	27OCT87
1126	F	TA3	40899	28MAY63	21NOV80
1677	M	BCK	26007	05NOV63	27MAR89
1441	F	FA2	27158	19NOV69	23MAR91
1421	M	TA2	33155	08JAN59	28FEB90
1119	M	TA1	26924	20JUN62	06SEP88
1834	M	BCK	26896	08FEB72	02JUL92
1777	M	PT3	109630	23SEP51	21JUN81
1663	M	BCK	26452	11JAN67	11AUG91
1106	M	PT2	89632	06NOV57	16AUG84
1103	F	FA1	23738	16FEB68	23JUL92
1477	M	FA2	28566	21MAR64	07MAR88
1476	F	TA2	34803	30MAY66	17MAR87
1379	M	ME3	42264	08AUG61	10JUN84
1104	M	SCP	17946	25APR63	10JUN91
1009	M	TA1	28880	02MAR59	26MAR92
1412	M	ME1	27799	18JUN56	05DEC91
1115	F	FA3	32699	22AUG60	29FEB80
1128	F	TA2	32777	23MAY65	20OCT90
1442	F	PT2	84536	05SEP66	12APR88

1417	M	NA2	52270	27JUN64	07MAR89
1478	M	PT2	84203	09AUG59	24OCT90
1673	M	BCK	25477	27FEB70	15JUL91
1839	F	NA1	43433	29NOV70	03JUL93
1347	M	TA3	40079	21SEP67	06SEP84
1423	F	ME2	35773	14MAY68	19AUG90
1200	F	ME1	27816	10JAN71	14AUG92
1970	F	FA1	22615	25SEP64	12MAR91
1521	M	ME3	41526	12APR63	13JUL88
1354	F	SCP	18335	29MAY71	16JUN92
1424	F	FA2	28978	04AUG69	11DEC89
1132	F	FA1	22413	30MAY72	22OCT93
1845	M	BCK	25996	20NOV59	22MAR80
1556	M	PT1	71349	22JUN64	11DEC91
1413	M	FA2	27435	16SEP65	02JAN90
1123	F	TA1	28407	31OCT72	05DEC92
1907	M	TA2	33329	15NOV60	06JUL87
1436	F	TA2	34475	11JUN64	12MAR87
1385	M	ME3	43900	16JAN62	01APR86
1432	F	ME2	35327	03NOV61	10FEB85
1111	M	NA1	40586	14JUL73	31OCT92
1116	F	FA1	22862	28SEP69	21MAR91
1352	M	NA2	53798	02DEC60	16OCT86
1555	F	FA2	27499	16MAR68	04JUL92
1038	F	TA1	26533	09NOV69	23NOV91
1420	M	ME3	43071	19FEB65	22JUL87
1561	M	TA2	34514	30NOV63	07OCT87
1434	F	FA2	28622	11JUL62	28OCT90
1414	M	FA1	23644	24MAR72	12APR92
1112	M	TA1	26905	29NOV64	07DEC92
1390	M	FA2	27761	19FEB65	23JUN91
1332	M	NA1	42178	17SEP70	04JUN91
1890	M	PT2	91908	20JUL51	25NOV79
1429	F	TA1	27939	28FEB60	07AUG92
1107	M	PT2	89977	09JUN54	10FEB79
1908	F	TA2	32995	10DEC69	23APR90
1830	F	PT2	84471	27MAY57	29JAN83
1882	M	ME3	41538	10JUL57	21NOV78
1050	M	ME2	35167	14JUL63	24AUG86
1425	F	FA1	23979	28DEC71	28FEB93
1928	M	PT2	89858	16SEP54	13JUL90
1480	F	TA3	39583	03SEP57	25MAR81
1100	M	BCK	25004	01DEC60	07MAY88
1995	F	ME1	28810	24AUG73	19SEP93
1135	F	FA2	27321	20SEP60	31MAR90
1415	M	FA2	28278	09MAR58	12FEB88
1076	M	PT1	66558	14OCT55	03OCT91
1426	F	TA2	32991	05DEC66	25JUN90
1564	F	SCP	18833	12APR62	01JUL92
1221	F	FA2	27896	22SEP67	04OCT91
1133	M	TA1	27701	13JUL66	12FEB92
1435	F	TA3	38808	12MAY59	08FEB80
1418	M	ME1	28005	29MAR57	06JAN92
1017	M	TA3	40858	28DEC57	16OCT81
1443	F	NA1	42274	17NOV68	29AUG91
1131	F	TA2	32575	26DEC71	19APR91

```
1427   F    TA2         34046  31OCT70  30JAN90
1036   F    TA3         39392  19MAY65  23OCT84
1130   F    FA1         23916  16MAY71  05JUN92
1127   F    TA2         33011  09NOV64  07DEC86
1433   F    FA3         32982  08JUL66  17JAN87
1431   F    FA3         33230  09JUN64  05APR88
1122   F    FA2         27956  01MAY63  27NOV88
1105   M    ME2         34805  01MAR62  13AUG90
;
```

PROCLIB.STAFF

```
data proclib.staff;
   input idnum $4. +3 lname $15. +2 fname $15. +2 city $15. +2
         state $2. +5 hphone $12.;
   datalines;
1919   ADAMS            GERALD           STAMFORD         CT     203/781-1255
1653   ALIBRANDI        MARIA            BRIDGEPORT       CT     203/675-7715
1400   ALHERTANI        ABDULLAH         NEW YORK         NY     212/586-0808
1350   ALVAREZ          MERCEDES         NEW YORK         NY     718/383-1549
1401   ALVAREZ          CARLOS           PATERSON         NJ     201/732-8787
1499   BAREFOOT         JOSEPH           PRINCETON        NJ     201/812-5665
1101   BAUCOM           WALTER           NEW YORK         NY     212/586-8060
1333   BANADYGA         JUSTIN           STAMFORD         CT     203/781-1777
1402   BLALOCK          RALPH            NEW YORK         NY     718/384-2849
1479   BALLETTI         MARIE            NEW YORK         NY     718/384-8816
1403   BOWDEN           EARL             BRIDGEPORT       CT     203/675-3434
1739   BRANCACCIO       JOSEPH           NEW YORK         NY     212/587-1247
1658   BREUHAUS         JEREMY           NEW YORK         NY     212/587-3622
1428   BRADY            CHRISTINE        STAMFORD         CT     203/781-1212
1782   BREWCZAK         JAKOB            STAMFORD         CT     203/781-0019
1244   BUCCI            ANTHONY          NEW YORK         NY     718/383-3334
1383   BURNETTE         THOMAS           NEW YORK         NY     718/384-3569
1574   CAHILL           MARSHALL         NEW YORK         NY     718/383-2338
1789   CARAWAY          DAVIS            NEW YORK         NY     212/587-9000
1404   COHEN            LEE              NEW YORK         NY     718/384-2946
1437   CARTER           DOROTHY          BRIDGEPORT       CT     203/675-4117
1639   CARTER-COHEN     KAREN            STAMFORD         CT     203/781-8839
1269   CASTON           FRANKLIN         STAMFORD         CT     203/781-3335
1065   COPAS            FREDERICO        NEW YORK         NY     718/384-5618
1876   CHIN             JACK             NEW YORK         NY     212/588-5634
1037   CHOW             JANE             STAMFORD         CT     203/781-8868
1129   COUNIHAN         BRENDA           NEW YORK         NY     718/383-2313
1988   COOPER           ANTHONY          NEW YORK         NY     212/587-1228
1405   DACKO            JASON            PATERSON         NJ     201/732-2323
1430   DABROWSKI        SANDRA           BRIDGEPORT       CT     203/675-1647
1983   DEAN             SHARON           NEW YORK         NY     718/384-1647
1134   DELGADO          MARIA            STAMFORD         CT     203/781-1528
1118   DENNIS           ROGER            NEW YORK         NY     718/383-1122
1438   DABBOUSSI        KAMILLA          STAMFORD         CT     203/781-2229
1125   DUNLAP           DONNA            NEW YORK         NY     718/383-2094
1475   ELGES            MARGARETE        NEW YORK         NY     718/383-2828
1117   EDGERTON         JOSHUA           NEW YORK         NY     212/588-1239
```

1935	FERNANDEZ	KATRINA	BRIDGEPORT	CT	203/675-2962
1124	FIELDS	DIANA	WHITE PLAINS	NY	914/455-2998
1422	FUJIHARA	KYOKO	PRINCETON	NJ	201/812-0902
1616	FUENTAS	CARLA	NEW YORK	NY	718/384-3329
1406	FOSTER	GERALD	BRIDGEPORT	CT	203/675-6363
1120	GARCIA	JACK	NEW YORK	NY	718/384-4930
1094	GOMEZ	ALAN	BRIDGEPORT	CT	203/675-7181
1389	GOLDSTEIN	LEVI	NEW YORK	NY	718/384-9326
1905	GRAHAM	ALVIN	NEW YORK	NY	212/586-8815
1407	GREGORSKI	DANIEL	MT. VERNON	NY	914/468-1616
1114	GREENWALD	JANICE	NEW YORK	NY	212/588-1092
1410	HARRIS	CHARLES	STAMFORD	CT	203/781-0937
1439	HASENHAUER	CHRISTINA	BRIDGEPORT	CT	203/675-4987
1409	HAVELKA	RAYMOND	STAMFORD	CT	203/781-9697
1408	HENDERSON	WILLIAM	PRINCETON	NJ	201/812-4789
1121	HERNANDEZ	ROBERTO	NEW YORK	NY	718/384-3313
1991	HOWARD	GRETCHEN	BRIDGEPORT	CT	203/675-0007
1102	HERMANN	JOACHIM	WHITE PLAINS	NY	914/455-0976
1356	HOWARD	MICHAEL	NEW YORK	NY	212/586-8411
1545	HERRERO	CLYDE	STAMFORD	CT	203/781-1119
1292	HUNTER	HELEN	BRIDGEPORT	CT	203/675-4830
1440	JACKSON	LAURA	STAMFORD	CT	203/781-0088
1368	JEPSEN	RONALD	STAMFORD	CT	203/781-8413
1369	JONSON	ANTHONY	NEW YORK	NY	212/587-5385
1411	JOHNSEN	JACK	PATERSON	NJ	201/732-3678
1113	JOHNSON	LESLIE	NEW YORK	NY	718/383-3003
1704	JONES	NATHAN	NEW YORK	NY	718/384-0049
1900	KING	WILLIAM	NEW YORK	NY	718/383-3698
1126	KIMANI	ANNE	NEW YORK	NY	212/586-1229
1677	KRAMER	JACKSON	BRIDGEPORT	CT	203/675-7432
1441	LAWRENCE	KATHY	PRINCETON	NJ	201/812-3337
1421	LEE	RUSSELL	MT. VERNON	NY	914/468-9143
1119	LI	JEFF	NEW YORK	NY	212/586-2344
1834	LEBLANC	RUSSELL	NEW YORK	NY	718/384-0040
1777	LUFKIN	ROY	NEW YORK	NY	718/383-4413
1663	MARKS	JOHN	NEW YORK	NY	212/587-7742
1106	MARSHBURN	JASPER	STAMFORD	CT	203/781-1457
1103	MCDANIEL	RONDA	NEW YORK	NY	212/586-0013
1477	MEYERS	PRESTON	BRIDGEPORT	CT	203/675-8125
1476	MONROE	JOYCE	STAMFORD	CT	203/781-2837
1379	MORGAN	ALFRED	STAMFORD	CT	203/781-2216
1104	MORGAN	CHRISTOPHER	NEW YORK	NY	718/383-9740
1009	MORGAN	GEORGE	NEW YORK	NY	212/586-7753
1412	MURPHEY	JOHN	PRINCETON	NJ	201/812-4414
1115	MURPHY	ALICE	NEW YORK	NY	718/384-1982
1128	NELSON	FELICIA	BRIDGEPORT	CT	203/675-1166
1442	NEWKIRK	SANDRA	PRINCETON	NJ	201/812-3331
1417	NEWKIRK	WILLIAM	PATERSON	NJ	201/732-6611
1478	NEWTON	JAMES	NEW YORK	NY	212/587-5549
1673	NICHOLLS	HENRY	STAMFORD	CT	203/781-7770
1839	NORRIS	DIANE	NEW YORK	NY	718/384-1767
1347	O'NEAL	BRYAN	NEW YORK	NY	718/384-0230
1423	OSWALD	LESLIE	MT. VERNON	NY	914/468-9171
1200	OVERMAN	MICHELLE	STAMFORD	CT	203/781-1835
1970	PARKER	ANNE	NEW YORK	NY	718/383-3895
1521	PARKER	JAY	NEW YORK	NY	212/587-7603

```
1354   PARKER          MARY            WHITE PLAINS     NY     914/455-2337
1424   PATTERSON       RENEE           NEW YORK         NY     212/587-8991
1132   PEARCE          CAROL           NEW YORK         NY     718/384-1986
1845   PEARSON         JAMES           NEW YORK         NY     718/384-2311
1556   PENNINGTON      MICHAEL         NEW YORK         NY     718/383-5681
1413   PETERS          RANDALL         PRINCETON        NJ     201/812-2478
1123   PETERSON        SUZANNE         NEW YORK         NY     718/383-0077
1907   PHELPS          WILLIAM         STAMFORD         CT     203/781-1118
1436   PORTER          SUSAN           NEW YORK         NY     718/383-5777
1385   RAYNOR          MILTON          BRIDGEPORT       CT     203/675-2846
1432   REED            MARILYN         MT. VERNON       NY     914/468-5454
1111   RHODES          JEREMY          PRINCETON        NJ     201/812-1837
1116   RICHARDS        CASEY           NEW YORK         NY     212/587-1224
1352   RIVERS          SIMON           NEW YORK         NY     718/383-3345
1555   RODRIGUEZ       JULIA           BRIDGEPORT       CT     203/675-2401
1038   RODRIGUEZ       MARIA           BRIDGEPORT       CT     203/675-2048
1420   ROUSE           JEREMY          PATERSON         NJ     201/732-9834
1561   SANDERS         RAYMOND         NEW YORK         NY     212/588-6615
1434   SANDERSON       EDITH           STAMFORD         CT     203/781-1333
1414   SANDERSON       NATHAN          BRIDGEPORT       CT     203/675-1715
1112   SANYERS         RANDY           NEW YORK         NY     718/384-4895
1390   SMART           JONATHAN        NEW YORK         NY     718/383-1141
1332   STEPHENSON      ADAM            BRIDGEPORT       CT     203/675-1497
1890   STEPHENSON      ROBERT          NEW YORK         NY     718/384-9874
1429   THOMPSON        ALICE           STAMFORD         CT     203/781-3857
1107   THOMPSON        WAYNE           NEW YORK         NY     718/384-3785
1908   TRENTON         MELISSA         NEW YORK         NY     212/586-6262
1830   TRIPP           KATHY           BRIDGEPORT       CT     203/675-2479
1882   TUCKER          ALAN            NEW YORK         NY     718/384-0216
1050   TUTTLE          THOMAS          WHITE PLAINS     NY     914/455-2119
1425   UNDERWOOD       JENNY           STAMFORD         CT     203/781-0978
1928   UPCHURCH        LARRY           WHITE PLAINS     NY     914/455-5009
1480   UPDIKE          THERESA         NEW YORK         NY     212/587-8729
1100   VANDEUSEN       RICHARD         NEW YORK         NY     212/586-2531
1995   VARNER          ELIZABETH       NEW YORK         NY     718/384-7113
1135   VEGA            ANNA            NEW YORK         NY     718/384-5913
1415   VEGA            FRANKLIN        NEW YORK         NY     718/384-2823
1076   VENTER          RANDALL         NEW YORK         NY     718/383-2321
1426   VICK            THERESA         PRINCETON        NJ     201/812-2424
1564   WALTERS         ANNE            NEW YORK         NY     212/587-3257
1221   WALTERS         DIANE           NEW YORK         NY     718/384-1918
1133   WANG            CHIN            NEW YORK         NY     212/587-1956
1435   WARD            ELAINE          NEW YORK         NY     718/383-4987
1418   WATSON          BERNARD         NEW YORK         NY     718/383-1298
1017   WELCH           DARIUS          NEW YORK         NY     212/586-5535
1443   WELLS           AGNES           STAMFORD         CT     203/781-5546
1131   WELLS           NADINE          NEW YORK         NY     718/383-1045
1427   WHALEY          CAROLYN         MT. VERNON       NY     914/468-4528
1036   WONG            LESLIE          NEW YORK         NY     212/587-2570
1130   WOOD            DEBORAH         NEW YORK         NY     212/587-0013
1127   WOOD            SANDRA          NEW YORK         NY     212/587-2881
1433   YANCEY          ROBIN           PRINCETON        NJ     201/812-1874
1431   YOUNG           DEBORAH         STAMFORD         CT     203/781-2987
1122   YOUNG           JOANN           NEW YORK         NY     718/384-2021
1105   YOUNG           LAWRENCE        NEW YORK         NY     718/384-0008
;
```

RADIO

This DATA step uses an INFILE statement to read data that is stored in an external file.

```
data radio;
   infile 'input-file' missover;
   input /(time1-time7) ($1. +1);
   listener=_n_;
run;
```

Here is the data that is stored in the external file:

```
967 32 f 5 3 5
7 5 5 5 7 0 0 0 8 7 0 0 8 0
781 30 f 2 3 5
5 0 0 0 5 0 0 0 4 7 5 0 0 0
859 39 f 1 0 5
1 0 0 0 1 0 0 0 0 0 0 0 0 0
859 40 f 6 1 5
7 5 0 5 7 0 0 0 0 0 0 5 0 0
467 37 m 2 3 1
1 5 5 5 5 4 4 8 8 0 0 0 0 0
220 35 f 3 1 7
7 0 0 0 7 0 0 0 7 0 0 0 0 0
833 42 m 2 2 4
7 0 0 0 7 5 4 7 4 0 1 4 4 0
967 39 f .5 1 7
7 0 0 0 7 7 0 0 0 0 0 0 8 0
677 28 m .5 .5 7
7 0 0 0 0 0 0 0 0 0 0 0 0 0
833 28 f 3 4 1
1 0 0 0 0 1 1 1 1 0 0 0 1 1
677 24 f 3 1 2
2 0 0 0 0 0 0 2 0 8 8 0 0 0
688 32 m 5 2 4
5 5 0 4 8 0 0 5 0 8 0 0 0 0
542 38 f 6 8 5
5 0 0 5 5 5 0 5 5 5 5 5 5 0
677 27 m 6 1 1
1 1 0 4 4 0 0 1 4 0 0 0 0 0
779 37 f 2.5 4 7
7 0 0 0 7 7 0 7 7 4 4 7 8 0
362 31 f 1 2 2
8 0 0 0 8 0 0 0 0 0 8 8 0 0
859 29 m 10 3 4
4 4 0 2 2 0 0 4 0 0 0 4 4 0
467 24 m 5 8 1
7 1 1 1 7 1 1 0 1 7 1 1 1 1
851 34 m 1 2 8
0 0 0 0 8 0 0 0 4 0 0 0 8 0
859 23 f 1 1 8
8 0 0 0 8 0 0 0 0 0 0 0 0 8
781 34 f 9 3 1
2 1 0 1 4 4 4 0 1 1 1 1 4 4
```

851 40 f 2 4 5
5 0 0 0 5 0 0 5 0 0 5 5 0 0
783 34 m 3 2 4
7 0 0 0 7 4 4 0 0 4 4 0 0 0
848 29 f 4 1.5 7
7 4 4 1 7 0 0 0 7 0 0 7 0 0
851 28 f 1 2 2
2 0 2 0 2 0 0 0 0 2 2 2 0 0
856 42 f 1.5 1 2
2 0 0 0 0 0 0 2 0 0 0 0 0 0
859 29 m .5 .5 5
5 0 0 0 1 0 0 0 0 0 8 8 5 0
833 29 m 1 3 2
2 0 0 0 2 2 0 0 4 2 0 2 0 0
859 23 f 10 3 1
1 5 0 8 8 1 4 0 1 1 1 1 1 4
781 37 f .5 2 7
7 0 0 0 1 0 0 0 1 7 0 1 0 0
833 31 f 5 4 1
1 0 0 0 1 0 0 0 4 0 4 0 0 0
942 23 f 4 2 1
1 0 0 0 1 0 1 0 1 1 0 0 0 0
848 33 f 5 4 1
1 1 0 1 1 0 0 0 1 1 1 0 0 0
222 33 f 2 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
851 45 f .5 1 8
8 0 0 0 8 0 0 0 0 0 8 0 0 0
848 27 f 2 4 1
1 0 0 0 1 1 0 0 4 1 1 1 1 1
781 38 m 2 2 1
5 0 0 0 1 0 0 0 0 0 1 1 0 0
222 27 f 3 1 2
2 0 2 0 2 2 0 0 2 0 0 0 0 0
467 34 f 2 2 1
1 0 0 0 0 1 0 1 0 0 0 0 1 0
833 27 f 8 8 1
7 0 1 0 7 4 0 0 1 1 1 4 1 0
677 49 f 1.5 0 8
8 0 8 0 8 0 0 0 0 0 0 0 0 0
849 43 m 1 4 1
1 0 0 0 4 0 0 0 4 0 1 0 0 0
467 28 m 2 1 7
7 0 0 0 7 0 0 7 0 0 1 0 0 0
732 29 f 1 0 2
2 0 0 0 2 0 0 0 0 0 0 0 0 0
851 31 m 2 2 2
2 5 0 6 0 0 8 0 2 2 8 2 0 0
779 42 f 8 2 2
7 2 0 2 7 0 0 0 0 0 0 0 2 0
493 40 m 1 3 3
3 0 0 0 5 3 0 5 5 0 0 0 1 1
859 30 m 1 0 7
7 0 0 0 7 0 0 0 0 0 0 0 0 0
833 36 m 4 2 5
7 5 0 5 0 5 0 0 7 0 0 0 5 0

```
467 30 f 1 4 1
0 0 0 0 1 0 6 0 0 1 1 1 0 6
859 32 f 3 5 2
2 2 2 2 2 2 6 6 2 2 2 2 2 6
851 43 f 8 1 5
7 5 5 5 0 0 0 4 0 0 0 0 0 0
848 29 f 3 5 1
7 0 0 0 7 1 0 0 1 1 1 1 1 0
833 25 f 2 4 5
7 0 0 0 5 7 0 0 7 5 0 0 5 0
783 33 f 8 3 8
8 0 8 0 7 0 0 0 8 0 5 4 0 5
222 26 f 10 2 1
1 1 0 1 1 0 0 0 3 1 1 0 0 0
222 23 f 3 2 2
2 2 2 2 7 0 0 2 2 0 0 0 0 0
859 50 f 1 5 4
7 0 0 0 7 0 0 5 4 4 4 7 0 0
833 26 f 3 2 1
1 0 0 1 1 0 0 5 5 0 1 0 0 0
467 29 m 7 2 1
1 1 1 1 1 0 0 1 1 1 0 0 0 0
859 35 m .5 2 2
7 0 0 0 2 0 0 7 5 0 0 4 0 0
833 33 f 3 3 6
7 0 0 0 6 8 0 8 0 0 0 8 6 0
221 36 f .5 1 5
0 7 0 0 0 7 0 0 7 0 0 7 7 0
220 32 f 2 4 5
5 0 5 0 5 5 5 0 5 5 5 5 5 5
684 19 f 2 4 2
0 2 0 2 0 0 0 0 0 2 2 0 0 0
493 55 f 1 0 5
5 0 0 5 0 0 0 0 7 0 0 0 0 0
221 27 m 1 1 7
7 0 0 0 0 0 0 0 5 0 0 0 5 0
684 19 f 0 .5 1
7 0 0 0 0 1 1 0 0 0 0 0 1 1
493 38 f .5 .5 5
0 8 0 0 5 0 0 0 5 0 0 0 0 0
221 26 f .5 2 1
0 1 0 0 0 1 0 0 5 5 5 1 0 0
684 18 m 1 .5 1
0 2 0 0 0 0 1 0 0 0 0 1 1 0
684 19 m 1 1 1
0 0 0 1 1 0 0 0 0 0 1 0 0 0
221 29 m .5 .5 5
0 0 0 0 0 5 5 0 0 0 0 0 5 5
683 18 f 2 4 8
0 0 0 0 8 0 0 0 8 8 8 0 0 0
966 23 f 1 2 1
1 5 5 5 1 0 0 0 0 1 0 0 1 0
493 25 f 3 5 7
7 0 0 0 7 2 0 0 7 0 2 7 7 0
683 18 f .5 .5 2
1 0 0 0 0 0 5 0 0 1 0 0 0 1
```

```
382 21 f 3 1 8
0 8 0 0 5 8 8 0 0 8 8 0 0 0
683 18 f 4 6 2
2 0 0 0 2 2 2 0 2 0 2 2 2 0
684 19 m .5 2 1
0 0 0 0 1 1 0 0 0 1 1 1 1 5
684 19 m 1.5 3.5 2
2 0 0 0 2 0 0 0 0 0 2 5 0 0
221 23 f 1 5 1
7 5 1 5 1 3 1 7 5 1 5 1 3 1
684 18 f 2 3 1
2 0 0 1 1 1 1 7 2 0 1 1 1 1
683 19 f 3 5 2
2 0 0 2 0 6 1 0 1 1 2 2 6 1
683 19 f 3 5 1
2 0 0 2 0 6 1 0 1 1 2 0 2 1
221 35 m 3 5 5
7 5 0 1 7 0 0 5 5 5 0 0 0 0
221 43 f 1 4 5
1 0 0 0 5 0 0 5 5 0 0 0 0 0
493 32 f 2 1 6
0 0 0 6 0 0 0 0 0 0 0 0 4 0
221 24 f 4 5 2
2 0 5 0 0 2 4 4 4 5 0 0 2 2
684 19 f 2 3 2
0 5 5 2 5 0 1 0 5 5 2 2 2 2
221 19 f 3 3 8
0 1 1 8 8 8 4 0 5 4 1 8 8 4
221 29 m 1 1 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5
221 21 m 1 1 1
1 0 0 0 0 0 5 1 0 0 0 0 0 5
683 20 f 1 2 2
0 0 0 0 2 0 0 0 2 0 0 0 0 0
493 54 f 1 1 5
7 0 0 5 0 0 0 0 0 0 5 0 0 0
493 45 m 4 6 5
7 0 0 0 7 5 0 0 5 5 5 5 5 5
850 44 m 2.5 1.5 7
7 0 7 0 4 7 5 0 5 4 3 0 0 4
220 33 m 5 3 5
1 5 0 5 1 0 0 0 0 0 0 0 5 5
684 20 f 1.5 3 1
1 0 0 0 1 0 1 0 1 0 0 1 1 0
966 63 m 3 5 3
5 4 7 5 4 5 0 5 0 0 5 5 4 0
683 21 f 4 6 1
0 1 0 1 1 1 1 0 1 1 1 1 1 1
493 23 f 5 2 5
7 5 0 4 0 0 0 0 1 1 1 1 1 0
493 32 f 8 8 5
7 5 0 0 7 0 5 5 5 0 0 7 5 5
942 33 f 7 2 5
0 5 5 4 7 0 0 0 0 0 0 7 8 0
493 34 f .5 1 5
5 0 0 0 5 0 0 0 0 0 6 0 0 0
```

```
382 40 f 2 2 5
5 0 0 0 5 0 0 5 0 0 5 0 0 0
362 27 f 0 3 8
0 0 0 0 0 0 0 0 0 0 0 0 8 0
542 36 f 3 3 7
7 0 0 0 7 1 0 0 0 7 1 1 0 0
966 39 f 3 6 5
7 0 0 0 7 5 0 0 7 0 5 0 5 0
849 32 m 1 .5 7
7 0 0 0 5 0 0 0 7 4 4 5 7 0
677 52 f 3 2 3
7 0 0 0 0 7 0 0 0 7 0 0 3 0
222 25 m 2 4 1
1 0 0 0 1 0 0 0 1 0 1 0 0 0
732 42 f 3 2 7
7 0 0 0 1 7 5 5 7 0 0 3 4 0
467 26 f 4 4 1
7 0 1 0 7 1 0 0 7 7 4 7 0 0
467 38 m 2.5 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
382 37 f 1.5 .5 7
7 0 0 0 7 0 0 0 3 0 0 0 3 0
856 45 f 3 3 7
7 0 0 0 7 5 0 0 7 7 4 0 0 0
677 33 m 3 2 7
7 0 0 4 7 0 0 0 7 0 0 0 0 0
490 27 f .5 1 2
2 0 0 0 2 0 0 0 2 0 2 0 0 0
362 27 f 1.5 2 2
2 0 0 0 1 0 4 0 1 0 0 0 4 4
783 25 f 2 1 1
1 0 0 0 1 7 0 0 0 0 1 1 1 0
546 30 f 8 3 1
1 1 1 1 1 0 0 1 0 5 5 0 0 0
677 30 f 2 0 1
1 0 0 0 0 1 0 0 0 0 0 0 0 1
221 35 f 2 2 1
1 0 0 0 1 0 1 0 1 1 1 0 0 0
966 32 f 6 1 7
7 1 1 1 7 4 0 1 7 1 8 8 4 0
222 28 f 1 5 4
7 0 0 0 4 0 0 4 4 4 4 0 0 0
467 29 f 5 3 4
4 5 5 5 1 4 4 5 1 1 1 1 4 4
467 32 m 3 4 1
1 0 1 0 4 0 0 0 4 0 0 0 1 0
966 30 m 1.5 1 7
7 0 0 0 7 5 0 7 0 0 0 0 5 0
967 38 m 14 4 7
7 7 7 7 7 0 4 8 0 0 0 0 4 0
490 28 m 8 1 1
7 1 1 1 1 0 0 7 0 0 8 0 0 0
833 30 f .5 1 6
6 0 0 0 6 0 0 0 0 6 0 0 6 0
851 40 m 1 0 7
7 5 5 5 7 0 0 0 0 0 0 0 0 0
```

```
859 27 f 2 5 2
6 0 0 0 2 0 0 0 0 0 0 2 2 2
851 22 f 3 5 2
7 0 2 0 2 2 0 0 2 0 8 0 2 0
967 38 f 1 1.5 7
7 0 0 0 7 5 0 7 4 0 0 7 5 0
856 34 f 1.5 1 1
0 1 0 0 0 1 0 0 4 0 0 0 0 0
222 33 m .1 .1 7
7 0 0 0 7 0 0 0 0 0 7 0 0 0
856 22 m .50 .25 1
0 1 0 0 1 0 0 0 0 0 0 0 0 0
677 30 f 2 2 4
1 0 4 0 4 0 0 0 4 0 0 0 0 0
859 25 m 2 3 7
0 0 0 0 0 7 0 0 7 0 2 0 0 1
833 35 m 2 6 7
7 0 0 0 7 1 1 0 4 7 4 7 1 1
677 35 m 10 4 1
1 1 1 1 1 8 6 8 1 0 0 8 8 8
848 29 f 5 3 8
8 0 0 0 8 8 0 0 0 8 8 8 0 0
688 26 m 3 1 1
1 1 7 1 1 7 0 0 0 8 8 0 0 0
490 41 m 2 2 5
5 0 0 0 0 0 5 5 0 0 0 0 0 5
493 35 m 4 4 7
7 5 0 5 7 0 0 7 7 7 7 0 0 0
677 27 m 15 11 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
848 27 f 3 5 1
1 1 0 0 1 1 0 0 1 1 1 1 0 0
362 30 f 1 0 1
1 0 0 0 7 5 0 0 0 0 0 0 0 0
783 29 f 1 1 4
4 0 0 0 4 0 0 0 4 0 0 0 4 0
467 39 f .5 2 4
7 0 4 0 4 4 0 0 4 4 4 4 4 4
677 27 m 2 2 7
7 0 0 0 7 0 0 7 7 0 0 7 0 0
221 23 f 2.5 1 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
677 29 f 1 1 7
0 0 0 0 7 0 0 0 7 0 0 0 0 0
783 32 m 1 2 5
4 5 5 5 4 2 0 0 0 0 3 2 2 0
833 25 f 1 0 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0
859 24 f 7 3 7
1 0 0 0 1 0 0 0 0 1 0 0 1 0
677 29 m 2 2 8
0 8 8 0 8 0 0 0 8 8 8 0 0 0
688 31 m 8 2 5
7 5 5 5 5 7 0 0 7 7 0 0 0 0
856 31 m 9 4 1
1 1 1 1 1 0 0 0 0 0 0 0 1 0
```

```
856 44 f 1 0 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
677 37 f 3 3 1
0 0 1 0 0 0 0 0 4 4 0 0 0 0
859 27 m 2 .5 2
2 2 2 2 2 2 2 2 0 0 0 0 0 2
781 30 f 10 4 2
2 0 0 0 2 0 2 0 0 0 0 0 0 2
362 27 m 12 4 3
3 1 1 1 1 3 3 3 0 0 0 0 3 0
362 33 f 2 4 1
1 0 0 0 7 0 0 7 1 1 1 1 1 0
222 26 f 8 1 1
1 1 1 1 0 0 0 1 0 0 0 0 0 0
779 37 f 6 3 1
1 1 1 1 1 0 0 1 1 0 0 0 1 0
467 32 f 1 1 2
2 0 0 0 0 0 0 0 2 0 0 2 0 0
859 23 m 1 1 1
1 0 0 0 1 1 0 1 0 0 0 0 1 1
781 33 f 1 .5 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
779 28 m 5 2 1
1 1 1 1 1 0 0 0 0 7 7 1 1 0
677 28 m 3 1 5
7 5 5 5 5 6 0 0 6 6 6 6 6 0
677 25 f 9 2 5
1 5 5 5 5 1 1 0 1 1 1 1 1 1
848 30 f 6 2 8
8 0 0 0 2 7 0 0 0 0 2 0 2 0
546 36 f 4 6 4
7 0 0 0 4 4 0 5 5 5 5 2 4 4
222 30 f 2 3 2
2 2 0 0 2 0 0 0 2 0 2 2 0 0
383 32 m 4 1 2
2 0 0 0 2 0 0 2 0 0 0 0 0 0
851 43 f 8 1 6
4 6 0 6 4 0 0 0 0 0 0 0 0 0
222 27 f 1 3 1
1 1 0 1 1 1 0 0 1 0 0 0 4 0
833 22 f 1.5 2 1
1 0 0 0 1 1 0 0 1 1 1 0 0 0
467 29 f 2 1 8
8 0 8 0 8 0 0 0 0 0 8 0 0 0
856 28 f 2 3 1
1 0 0 0 1 0 0 0 1 0 0 1 0 0
580 31 f 2.5 2.5 6
6 6 6 6 6 6 6 6 1 1 1 1 6 6
688 39 f 8 8 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3
677 37 f 1.5 .5 1
6 1 1 1 6 6 0 0 1 1 6 6 6 0
859 38 m 3 6 3
7 0 0 0 7 3 0 0 3 0 3 0 0 0
677 25 f 7 1 1
0 1 1 1 2 0 0 0 1 2 1 1 1 0
```

```
848 36 f 7 1 1
0 1 0 1 1 0 0 0 0 0 0 1 1 0
781 31 f 2 4 1
1 0 0 0 1 1 0 1 1 1 1 1 0 0
781 40 f 2 2 8
8 0 0 8 8 0 0 0 0 0 8 8 0 0
677 25 f 3 5 1
1 6 1 6 6 3 0 0 2 2 1 1 1 1
779 33 f 3 2 1
1 0 1 0 0 0 1 0 1 0 0 0 1 0
677 25 m 7 1.5 1
1 1 0 1 1 0 0 0 0 0 1 0 0 0
362 35 f .5 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
677 41 f 6 2 7
7 7 0 7 7 0 0 0 0 0 8 0 0 0
677 24 m 5 1 5
1 5 0 5 0 0 0 0 1 0 0 0 0 0
833 29 f .5 0 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
362 30 f 1 1 1
1 0 0 0 1 0 0 0 1 0 0 0 0 0
850 26 f 6 12 6
6 0 0 0 2 2 2 6 6 6 0 0 6 6
467 25 f 2 3 1
1 0 0 6 1 1 0 0 0 0 1 1 1 1
967 29 f 1 2 7
7 0 0 0 7 0 0 7 7 0 0 0 0 0
833 31 f 1 1 7
7 0 7 0 7 3 0 0 3 3 0 0 0 0
859 40 f 7 1 5
1 5 0 5 5 1 0 0 1 0 0 0 0 0
848 31 m 1 2 1
1 0 0 0 1 1 0 0 4 4 1 4 0 0
222 32 f 2 3 3
3 0 0 0 0 7 0 0 3 0 8 0 0 0
783 33 f 2 0 4
7 0 0 0 7 0 0 0 4 0 4 0 0 0
856 28 f 8 4 2
0 2 0 2 2 0 0 0 2 0 2 0 4 0
781 30 f 3 5 1
1 1 1 1 1 1 0 0 1 1 1 1 1 0
850 25 f 6 3 1
7 5 0 5 7 1 0 0 7 0 1 0 1 0
580 33 f 2.5 4 2
2 0 0 0 2 0 0 0 0 0 8 8 0 0
677 38 f 3 3 1
1 0 0 0 1 0 1 1 1 0 1 0 0 4
677 26 f 2 2 1
1 0 1 0 1 0 0 0 1 1 1 0 0 0
467 52 f 3 2 2
2 6 6 6 6 2 0 0 2 2 2 2 0 0
542 31 f 1 3 1
1 0 1 0 1 0 0 0 1 1 1 1 1 0
859 50 f 9 3 6
6 6 6 6 6 6 6 6 6 3 3 3 6 6
```

```
779 26 f 1 2 1
7 0 1 0 1 1 4 1 4 1 1 1 4 4
779 36 m 1.5 2 4
1 4 0 4 4 0 0 4 4 4 4 0 0 0
222 31 f 0 3 7
1 0 0 0 7 0 0 0 0 0 0 0 0 0
362 27 f 1 1 1
1 0 1 0 1 4 0 4 4 1 0 4 4 0
967 32 f 3 2 7
7 0 0 0 7 0 0 0 1 0 0 1 0 0
362 29 f 10 2 2
2 2 2 2 2 2 2 2 2 2 2 7 0 0
677 27 f 3 4 1
0 5 1 1 0 5 0 0 0 1 1 1 0 0
546 32 m 5 .5 8
8 0 0 0 8 0 0 0 8 0 0 0 0 0
688 38 m 2 3 2
2 0 0 0 2 0 0 0 2 0 0 0 1 0
362 28 f 1 1 1
1 0 0 0 1 1 0 4 0 0 0 0 4 0
851 32 f .5 2 4
5 0 0 0 4 0 0 0 0 0 0 0 2 0
967 43 f 2 2 1
1 0 0 0 1 0 0 1 7 0 0 0 1 0
467 44 f 10 4 6
7 6 0 6 6 0 6 0 0 0 0 0 0 6
467 23 f 5 3 1
0 2 1 2 1 0 0 0 1 1 1 1 1 1
783 30 f 1 .5 1
1 0 0 0 1 0 0 0 0 0 0 7 0 0
677 29 f 3 1 2
2 2 2 2 2 0 0 0 0 0 0 0 0 0
859 26 f 9.5 1.5 2
2 2 2 2 2 0 0 2 2 0 0 0 0 0
222 28 f 3 0 2
2 0 0 0 2 0 0 0 0 0 2 0 0 0
966 37 m 2 1 1
7 1 1 1 7 0 0 0 7 0 0 0 0 0
859 31 f 10 10 1
0 1 1 1 1 0 0 0 1 1 0 0 1 0
781 27 f 2 1 2
2 0 0 0 1 0 0 0 4 0 0 0 0 0
677 31 f .5 .5 6
7 0 0 0 0 0 0 0 6 0 0 0 0 0
848 28 f 5 1 2
2 2 0 2 0 0 0 0 2 0 0 0 0 0
781 24 f 3 3 6
1 6 6 6 1 6 0 0 0 0 1 0 1 1
856 27 f 1.5 1 6
2 6 6 6 2 5 0 2 0 0 5 2 0 0
382 30 m 1 2 7
7 0 0 0 7 0 4 7 0 0 0 7 4 4
848 25 f 9 3 1
7 1 1 5 1 0 0 0 1 1 1 1 1 0
382 30 m 1 2 4
7 0 0 0 7 0 4 7 0 0 0 7 4 4
```

688 40 m 2 3 1
1 0 0 0 1 3 1 0 5 0 4 4 7 1
856 40 f .5 5 5
3 0 0 0 3 0 0 0 0 0 5 5 0 0
966 25 f 2 .5 2
1 0 0 0 2 6 0 0 4 0 0 0 0 0
859 30 f 2 4 2
2 0 0 0 0 2 0 0 0 0 2 0 0 0
849 29 m 10 1 5
7 5 5 5 7 5 5 0 0 0 0 0 7 0
781 28 m 1.5 3 4
1 0 0 0 1 4 4 0 4 4 1 1 4 0
467 35 f 4 2 6
7 6 7 6 6 7 6 7 7 7 7 7 7 6
222 32 f 10 5 1
1 1 0 1 1 0 0 1 1 1 0 0 1 0
677 32 f 1 0 1
1 0 1 0 0 0 0 0 0 0 0 0 0 0
222 54 f 21 4 3
5 0 0 0 7 0 0 7 0 0 0 0 0 0
677 30 m 4 6 1
7 0 0 0 0 1 1 1 7 1 1 0 8 1
683 29 f 1 2 8
8 0 0 0 8 0 0 0 0 8 8 0 0 0
467 38 m 3 5 1
1 0 0 0 1 0 0 1 1 0 0 0 0 0
781 29 f 2 3 8
8 0 0 0 8 8 0 0 8 8 0 8 8 0
781 30 f 1 0 5
5 0 0 0 0 5 0 0 0 0 0 0 0 0
783 40 f 1.5 3 1
1 0 0 0 1 4 0 0 1 1 1 0 0 0
851 30 f 1 1 6
6 0 0 0 6 0 0 0 6 0 0 6 0 0
851 40 f 1 1 5
5 0 0 0 5 0 0 0 0 1 0 0 0 0
779 40 f 1 0 2
2 0 0 0 2 0 0 0 0 0 0 0 0 0
467 37 f 4 8 1
1 0 0 0 1 0 3 0 3 1 1 1 0 0
859 37 f 4 3 3
0 3 7 0 0 7 0 0 0 7 8 3 7 0
781 26 f 4 1 2
2 2 0 2 1 0 0 0 2 0 0 0 0 0
859 23 f 8 3 3
3 2 0 2 3 0 0 0 1 0 0 3 0 0
967 31 f .5 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0
851 38 m 4 2 5
7 5 0 5 4 0 4 7 7 0 4 0 8 0
467 30 m 2 1 2
2 2 0 2 0 0 0 0 2 0 2 0 0 0
848 33 f 2 2 7
7 0 0 0 0 7 0 7 7 0 0 0 7 0
688 35 f 5 8 3
2 2 2 2 2 0 0 3 3 3 3 3 0 0

```
467 27 f 2 3 1
1 0 1 0 0 1 0 0 1 1 1 0 0 0
783 42 f 3 1 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0
687 40 m 1.5 2 1
7 0 0 0 1 1 0 0 1 0 7 0 1 0
779 30 f 4 8 7
7 0 0 0 7 0 6 7 4 2 2 0 0 6
222 34 f 9 0 8
8 2 0 2 8 0 0 0 0 0 0 0 0 0
467 28 m 3 1 2
2 0 0 0 2 2 0 0 0 2 2 0 0 0
222 28 f 8 4 2
1 2 1 2 2 0 0 1 2 2 0 0 2 0
542 35 m 2 3 2
6 0 7 0 7 0 7 0 0 0 2 2 0 0
677 31 m 12 4 3
7 3 0 3 3 4 0 0 4 4 4 0 0 0
783 45 f 1.5 2 6
6 0 0 0 6 0 0 6 6 0 0 0 0 0
942 34 f 1 .5 4
4 0 0 0 1 0 0 0 0 0 2 0 0 0
222 30 f 8 4 1
1 1 1 1 1 0 0 0 1 1 0 0 0 0
967 38 f 1.5 2 7
7 0 0 0 7 0 0 7 1 1 1 1 0 0
783 37 f 2 1 1
6 6 1 1 6 6 0 0 6 1 1 1 6 0
467 31 f 1.5 2 2
2 0 7 0 7 0 0 7 7 0 0 0 7 0
859 48 f 3 0 7
7 0 0 0 0 0 0 0 0 7 0 0 0 0
490 35 f 1 1 7
7 0 0 0 7 0 0 0 0 0 0 0 8 0
222 27 f 3 2 3
8 0 0 0 3 8 0 3 3 0 0 0 0 0
382 36 m 3 2 4
7 0 5 4 7 4 4 0 7 7 4 7 0 4
859 37 f 1 1 2
7 0 0 0 0 2 0 2 2 0 0 0 0 2
856 29 f 3 1 1
1 0 0 0 1 1 1 1 0 0 1 1 0 1
542 32 m 3 3 7
7 0 0 0 0 7 7 7 0 0 0 0 7 7
783 31 m 1 1 1
1 0 0 0 1 0 0 0 1 1 1 0 0 0
833 35 m 1 1 1
5 4 1 5 1 0 0 1 1 0 0 0 0 0
782 38 m 30 8 5
7 5 5 5 5 0 0 4 4 4 4 4 0 0
222 33 m 3 3 1
1 1 1 1 1 1 1 1 4 1 1 1 1 1
467 24 f 2 4 1
0 0 1 0 1 0 0 0 1 1 1 0 0 0
467 34 f 1 1 1
1 0 0 0 1 0 0 1 1 0 0 0 0 0
```

781 53 f 2 1 5
5 0 0 0 5 5 0 0 0 0 5 5 5 0
222 30 m 2 5 3
6 3 3 3 6 0 0 0 3 3 3 3 0 0
688 26 f 2 2 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0
222 29 m 8 5 1
1 6 0 6 1 0 0 1 1 1 1 0 0 0
783 33 m 1 2 7
7 0 0 0 7 0 0 0 7 0 0 0 7 0
781 39 m 1.5 2.5 2
2 0 2 0 2 0 0 0 2 2 2 0 0 0
850 22 f 2 1 1
1 0 0 0 1 1 1 0 5 0 0 1 0 0
493 36 f 1 0 5
0 0 0 0 7 0 0 0 0 0 0 0 0 0
967 46 f 2 4 7
7 5 0 5 7 0 0 0 4 7 4 0 0 0
856 41 m 2 2 4
7 4 0 0 7 4 0 4 0 0 0 7 0 0
546 25 m 5 5 8
8 8 0 0 0 0 0 0 0 0 0 0 0 0
222 27 f 4 4 3
2 2 2 3 7 7 0 2 2 2 3 3 3 0
688 23 m 9 3 3
3 3 3 3 3 7 0 0 3 0 0 0 0 0
849 26 m .5 .5 8
8 0 0 0 8 0 0 0 0 8 0 0 0 0
783 29 f 3 3 1
1 0 0 0 4 0 0 4 1 0 1 0 0 0
856 34 f 1.5 2 1
7 0 0 0 7 0 0 7 4 0 0 7 0 0
966 33 m 3 5 4
7 0 0 0 7 4 5 0 7 0 0 7 4 4
493 34 f 2 5 1
1 0 0 0 1 0 0 0 7 0 1 1 8 0
467 29 m 2 4 2
2 0 0 0 2 0 0 2 2 2 2 2 2 2
677 28 f 1 4 1
1 1 1 1 1 0 0 0 1 0 1 0 0 0
781 27 m 2 2 1
1 0 1 0 4 2 4 0 2 2 1 0 1 4
467 24 m 4 4 1
7 1 0 1 1 1 0 7 1 0 0 0 0 0
859 26 m 5 5 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
848 27 m 7 2 5
7 5 0 5 4 5 0 0 0 7 4 4 0 4
677 25 f 1 2 8
8 0 0 0 0 5 0 0 8 0 0 0 2 0
222 26 f 3.5 0 2
2 0 0 0 2 0 0 0 0 0 0 0 0 0
833 32 m 1 2 1
1 0 0 0 1 0 0 0 5 0 1 0 0 0
781 28 m 2 .5 7
7 0 0 0 7 0 0 0 4 0 0 0 0 0

```
783 28 f 1 1 1
1 0 0 0 1 0 0 0 0 0 1 1 0 0
222 28 f 5 5 2
2 6 6 2 2 0 0 0 2 2 0 0 2 2
851 33 m 4 5 3
1 0 0 0 7 3 0 3 3 3 3 3 7 5
859 39 m 2 1 1
1 0 0 0 1 0 0 0 0 0 0 1 0 0
848 45 m 2 2 7
7 0 0 0 7 0 0 0 7 0 0 0 0 0
467 37 m 2 2 7
7 0 0 0 0 7 0 0 0 7 0 0 7 0
859 32 m .25 .25 1
1 0 0 0 0 0 0 0 1 0 0 0 0 0
```

SALES

```
data sales;
   input Region $ CitySize $ Population Product $ SaleType $ Units NetSales;
   cards;
NC S 25000 A100 R 150 3750.00
NC M  125000 A100 R 350 8650.00
NC L 837000 A100 R 800 20000.00
NC S 25000 A100 W 150 3000.00
NC M 125000 A100 W 350 7000.00
NC M 625000 A100 W 750 15000.00
TX M 227000 A100 W 350 7250.00
TX L 5000 A100 W 750 5000.00
;
```

Appendix 2

ICU License

ICU License - ICU 1.8.1 and later

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Third-Party Software Licenses

This section contains third-party software notices and/or additional terms for licensed third-party software components included within ICU libraries.

1. Unicode Data Files and Software

EXHIBIT 1

UNICODE, INC. LICENSE AGREEMENT - DATA FILES AND SOFTWARE

Unicode Data Files include all data files under the directories http://www.unicode.org/Public/, http://www.unicode.org/reports/, and http://www.unicode.org/cldr/data/. Unicode Data Files do not include PDF online code charts under the directory http://www.unicode.org/Public/. Software includes any source code published in the Unicode Standard or under the directories http://www.unicode.org/Public/, http://www.unicode.org/reports/, and http://www.unicode.org/cldr/data/.

NOTICE TO USER: Carefully read the following legal agreement. BY DOWNLOADING, INSTALLING, COPYING OR OTHERWISE USING UNICODE INC.'S DATA FILES ("DATA FILES"), AND/OR SOFTWARE ("SOFTWARE"), YOU UNEQUIVOCALLY ACCEPT, AND AGREE TO BE BOUND BY, ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE, DO NOT DOWNLOAD, INSTALL, COPY, DISTRIBUTE OR USE THE DATA FILES OR SOFTWARE.

COPYRIGHT AND PERMISSION NOTICE

Copyright © 1991-2014 Unicode, Inc. All rights reserved. Distributed under the Terms of Use in http://www.unicode.org/copyright.html.

Permission is hereby granted, free of charge, to any person obtaining a copy of the Unicode data files and any associated documentation (the "Data Files") or Unicode software and any associated documentation (the "Software") to deal in the Data Files or Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Data Files or Software, and to permit persons to whom the Data Files or Software are furnished to do so, provided that (a) the above copyright notice(s) and this permission notice appear with all copies of the Data Files or Software, (b) both the above copyright notice(s) and this permission notice appear in associated documentation, and (c) there is clear notice in each modified Data File or in the Software as well as in the documentation associated with the Data File(s) or Software that the data or software has been modified.

THE DATA FILES AND SOFTWARE ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR

CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE DATA FILES OR SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in these Data Files or Software without prior written authorization of the copyright holder.

Unicode and the Unicode logo are trademarks of Unicode, Inc. in the United States and other countries. All third party trademarks referenced herein are the property of their respective owners.

2. Chinese/Japanese Word Break Dictionary Data (cjdict.txt)

```
#  The Google Chrome software developed by Google is licensed under
#  the BSD license. Other software included in this distribution is provided
#  under other licenses, as set forth below.
#
#    The BSD License
#    http://opensource.org/licenses/bsd-license.php
#    Copyright (C) 2006-2008, Google Inc.
#
#    All rights reserved.
#
#    Redistribution and use in source and binary forms, with or without
#  modification, are permitted provided that the following conditions are met:
#
#    Redistributions of source code must retain the above copyright notice, this
#  list of conditions and the following disclaimer.
#    Redistributions in binary form must reproduce the above copyright notice,
#  this list of conditions and the following disclaimer in the documentation
#  and/or other materials provided with the distribution.
#    Neither the name of Google Inc. nor the names of its contributors may be
#  used to endorse or promote products derived from this software without
#  specific prior written permission.
#
#
#  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
#  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
#  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
#  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
#  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
#  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
#  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
#  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
#  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
#  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
#  THE POSSIBILITY OF SUCH DAMAGE.
#
#
#    The word list in cjdict.txt are generated by combining three word lists
#  listed below with further processing for compound word breaking. The
#  frequency is generated with an iterative training against Google
```

```
#  web corpora.
#
#    * Libtabe (Chinese)
#      - https://sourceforge.net/project/?group_id=1519
#      - Its license terms and conditions are shown below.
#
#    * IPADIC (Japanese)
#      - http://chasen.aist-nara.ac.jp/chasen/distribution.html
#      - Its license terms and conditions are shown below.
#
#    ---------COPYING.libtabe ---- BEGIN--------------------
#
#    /*
#     * Copyrighy (c) 1999 TaBE Project.
#     * Copyright (c) 1999 Pai-Hsiang Hsiao.
#     * All rights reserved.
#     *
#     * Redistribution and use in source and binary forms, with or without
#     * modification, are permitted provided that the following conditions
#     * are met:
#     *
#     * . Redistributions of source code must retain the above copyright
#     *   notice, this list of conditions and the following disclaimer.
#     * . Redistributions in binary form must reproduce the above copyright
#     *   notice, this list of conditions and the following disclaimer in
#     *   the documentation and/or other materials provided with the
#     *   distribution.
#     * . Neither the name of the TaBE Project nor the names of its
#     *   contributors may be used to endorse or promote products derived
#     *   from this software without specific prior written permission.
#     *
#     * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#     * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#     * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
#     * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
#     * REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
#     * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
#     * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
#     * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
#     * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
#     * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
#     * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
#     * OF THE POSSIBILITY OF SUCH DAMAGE.
#     */
#
#    /*
#     * Copyright (c) 1999 Computer Systems and Communication Lab,
#     *                    Institute of Information Science, Academia Sinica.
#     * All rights reserved.
#     *
#     * Redistribution and use in source and binary forms, with or without
#     * modification, are permitted provided that the following conditions
#     * are met:
#     *
#     * . Redistributions of source code must retain the above copyright
#     *   notice, this list of conditions and the following disclaimer.
```

```
#     * . Redistributions in binary form must reproduce the above copyright
#     *   notice, this list of conditions and the following disclaimer in
#     *   the documentation and/or other materials provided with the
#     *   distribution.
#     * . Neither the name of the Computer Systems and Communication Lab
#     *   nor the names of its contributors may be used to endorse or
#     *   promote products derived from this software without specific
#     *   prior written permission.
#     *
#     * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#     * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#     * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
#     * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
#     * REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
#     * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
#     * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
#     * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
#     * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
#     * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
#     * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
#     * OF THE POSSIBILITY OF SUCH DAMAGE.
#     */
#
#    Copyright 1996 Chih-Hao Tsai @ Beckman Institute, University of Illinois
#    c-tsai4@uiuc.edu  http://casper.beckman.uiuc.edu/~c-tsai4
#
#    ---------------COPYING.libtabe-----END-----------------------------------
#
#
#    ---------------COPYING.ipadic-----BEGIN-----------------------------------
#
#    Copyright 2000, 2001, 2002, 2003 Nara Institute of Science
#    and Technology.  All Rights Reserved.
#
#    Use, reproduction, and distribution of this software is permitted.
#    Any copy of this software, whether in its original form or modified,
#    must include both the above copyright notice and the following
#    paragraphs.
#
#    Nara Institute of Science and Technology (NAIST),
#    the copyright holders, disclaims all warranties with regard to this
#    software, including all implied warranties of merchantability and
#    fitness, in no event shall NAIST be liable for
#    any special, indirect or consequential damages or any damages
#    whatsoever resulting from loss of use, data or profits, whether in an
#    action of contract, negligence or other tortuous action, arising out
#    of or in connection with the use or performance of this software.
#
#    A large portion of the dictionary entries
#    originate from ICOT Free Software.  The following conditions for ICOT
#    Free Software applies to the current dictionary as well.
#
#    Each User may also freely distribute the Program, whether in its
#    original form or modified, to any third party or parties, PROVIDED
#    that the provisions of Section 3 ("NO WARRANTY") will ALWAYS appear
#    on, or be attached to, the Program, which is distributed substantially
```

```
#    in the same form as set out herein and that such intended
#    distribution, if actually made, will neither violate or otherwise
#    contravene any of the laws and regulations of the countries having
#    jurisdiction over the User or the intended distribution itself.
#
#    NO WARRANTY
#
#    The program was produced on an experimental basis in the course of the
#    research and development conducted during the project and is provided
#    to users as so produced on an experimental basis.  Accordingly, the
#    program is provided without any warranty whatsoever, whether express,
#    implied, statutory or otherwise.  The term "warranty" used herein
#    includes, but is not limited to, any warranty of the quality,
#    performance, merchantability and fitness for a particular purpose of
#    the program and the nonexistence of any infringement or violation of
#    any right of any third party.
#
#    Each user of the program will agree and understand, and be deemed to
#    have agreed and understood, that there is no warranty whatsoever for
#    the program and, accordingly, the entire risk arising from or
#    otherwise connected with the program is assumed by the user.
#
#    Therefore, neither ICOT, the copyright holder, or any other
#    organization that participated in or was otherwise related to the
#    development of the program and their respective officials, directors,
#    officers and other employees shall be held liable for any and all
#    damages, including, without limitation, general, special, incidental
#    and consequential damages, arising out of or otherwise in connection
#    with the use or inability to use the program or any product, material
#    or result produced or otherwise obtained by using the program,
#    regardless of whether they have been advised of, or otherwise had
#    knowledge of, the possibility of such damages at any time during the
#    project or thereafter.  Each user will be deemed to have agreed to the
#    foregoing by his or her commencement of use of the program.  The term
#    "use" as used herein includes, but is not limited to, the use,
#    modification, copying and distribution of the program and the
#    production of secondary products from the program.
#
#    In the case where the program, whether in its original form or
#    modified, was distributed or delivered to or received by a user from
#    any person, organization or entity other than ICOT, unless it makes or
#    grants independently of ICOT any specific warranty to the user in
#    writing, such person, organization or entity, will also be exempted
#    from and not be held liable to the user for any such damages as noted
#    above as far as the program is concerned.
#
#    ---------------COPYING.ipadic-----END-----------------------------------
```

Lao Word Break Dictionary Data (laodict.txt)

```
#    Copyright (c) 2013 International Business Machines Corporation
#    and others. All Rights Reserved.
#
#    Project:    http://code.google.com/p/lao-dictionary/
#    Dictionary: http://lao-dictionary.googlecode.com/git/Lao-Dictionary.txt
```

```
#    License:    http://lao-dictionary.googlecode.com/git/Lao-Dictionary-LICENSE.txt
#                (copied below)
#
#    This file is derived from the above dictionary, with slight modifications.
#    --------------------------------------------------------------------------------
#    Copyright (C) 2013 Brian Eugene Wilson, Robert Martin Campbell.
#    All rights reserved.
#
#    Redistribution and use in source and binary forms, with or without modification,
#    are permitted provided that the following conditions are met:
#
#        Redistributions of source code must retain the above copyright notice, this
#        list of conditions and the following disclaimer. Redistributions in binary
#        form must reproduce the above copyright notice, this list of conditions and
#        the following disclaimer in the documentation and/or other materials
#        provided with the distribution.
#
#    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
#    ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
#    WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
#    DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
#    ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
#    (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
#    LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
#    ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
#    (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
#    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#    --------------------------------------------------------------------------------
```

3. Time Zone Database

ICU uses the public domain data and code derived from Time Zone Database for its time zone support. The ownership of the TZ database is explained in BCP 175: Procedure for Maintaining the Time Zone Database section 7.

```
7.  Database Ownership

    The TZ database itself is not an IETF Contribution or an IETF
    document.  Rather it is a pre-existing and regularly updated work
    that is in the public domain, and is intended to remain in the public
    domain.  Therefore, BCPs 78 [RFC5378] and 79 [RFC3979] do not apply
    to the TZ Database or contributions that individuals make to it.
    Should any claims be made and substantiated against the TZ Database,
    the organization that is providing the IANA Considerations defined in
    this RFC, under the memorandum of understanding with the IETF,
    currently ICANN, may act in accordance with all competent court
    orders.  No ownership claims will be made by ICANN or the IETF Trust
    on the database or the code.  Any person making a contribution to the
    database or code waives all rights to future claims in that
    contribution or in the TZ Database.
```

Recommended Reading

Here is the recommended reading list for this title:

- *Carpenter's Complete Guide to the SAS® REPORT Procedure*
- *Learning SAS® by Example: A Programmer's Guide*
- *Little SAS® Book: A Primer, Fifth Edition*
- *Output Delivery System: The Basics and Beyond*
- *PROC REPORT by Example: Techniques for Building Professional Reports Using SAS*
- *PROC SQL: Beyond the Basics Using SAS,® Second Edition*
- *PROC TABULATE by Example*
- *SAS Viya Component Objects: Reference*
- *SAS Viya Data Set Options: Reference*
- *SAS Viya: DS2 Language Reference*
- *SAS Viya Formats and Informats: Reference*
- *SAS Viya Functions and CALL Routines: Reference*
- *SAS Viya SQL Procedure User's Guide*
- *SAS Viya Statements: Reference*
- *SAS Viya System Options: Reference*
- SAS offers instructor-led training and self-paced e-learning courses to help you get started with SAS Foundation. For more information about the courses available, see support.sas.com/training.

For a complete list of SAS publications, go to 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

Index

D

E

F

G

H

Q

R

S

T

U

V

W

X