# Contents

Early Adopter Software ......................................................... vii

## PART 1 Procedures 1

### Chapter 1 • APPEND Procedure ........................................ 3
  Overview: APPEND Procedure ........................................... 3
  Concepts: APPEND Procedure ........................................... 4
  Syntax: APPEND Procedure ............................................. 6
  Using the APPEND Procedure ........................................... 8
  Example: Concatenating a CAS Table to a SAS Data Set ............ 8

### Chapter 2 • CONTENTS Procedure ........................................ 13
  Overview: CONTENTS Procedure ....................................... 13
  Concepts: CONTENTS Procedure ....................................... 15
  Syntax: CONTENTS Procedure ......................................... 15
  Results: CONTENTS Procedure ....................................... 20
  Examples: CONTENTS Procedure ..................................... 25

### Chapter 3 • COPY Procedure ............................................. 35
  Overview: COPY Procedure ........................................... 35
  Concepts ................................................................. 35
  Syntax: COPY Procedure ............................................. 37
  Using the COPY Procedure ........................................... 41
  Example: Copy a SAS Data Set to a CAS Table .................... 41

### Chapter 4 • DATASETS Procedure ........................................ 43
  Overview: DATASETS Procedure ..................................... 43
  Concepts: DATASETS Procedure .................................... 44
  Syntax: DATASETS Procedure ....................................... 44
  Results: DATASETS Procedure .................................... 61

### Chapter 5 • DELETE Procedure ........................................... 69
  Overview: DELETE Procedure ........................................ 69
  Concepts: DELETE Procedure ....................................... 69
  Syntax: DELETE Procedure ........................................ 69
  Examples: DELETE Procedure ....................................... 70

### Chapter 6 • EXPORT Procedure .......................................... 73
  Overview: Export Procedure .......................................... 73
  Syntax: EXPORT Procedure ........................................ 74
  Examples: EXPORT Procedure ...................................... 79

### Chapter 7 • FORMAT Procedure .......................................... 95
  Overview: FORMAT Procedure ......................................... 96
  Concepts: FORMAT Procedure ...................................... 97
  Syntax: FORMAT Procedure ........................................ 102
  Specifying Values or Ranges ........................................ 139
### Chapter 8 • HTTP Procedure
- Overview: HTTP Procedure .......................................................... 175
- Syntax: HTTP Procedure ............................................................... 176
- Using Hypertext Transfer Protocol Secure (HTTPS) ................. 182
- Using Authentication Other Than Basic ................................. 182
- Wire Logging .............................................................................. 183
- Using Encodings with PROC HTTP .................................. 183
- PROC HTTP Macro Variables .............................................. 183
- Examples: HTTP Procedure ....................................................... 184

### Chapter 9 • IMPORT Procedure
- Overview: IMPORT Procedure ...................................................... 193
- Syntax: IMPORT Procedure ............................................................ 195
- Examples: IMPORT Procedure ..................................................... 202

### Chapter 10 • JAVAINFO Procedure
- Overview: JAVAINFO Procedure ............................................... 217
- Syntax: JAVAINFO Procedure ...................................................... 217

### Chapter 11 • OPTIONS Procedure
- Overview: OPTIONS Procedure .................................................... 219
- Syntax: OPTIONS Procedure .......................................................... 220
- Displaying a List of System Options .................................. 224
- Displaying Information about One or More Options ............. 225
- Displaying Information about System Option Groups .......... 227
- Displaying Restricted Options .................................................. 229
- Results: OPTIONS Procedure .................................................... 230
- Examples: OPTIONS Procedure ................................................. 230

### Chapter 12 • PRINT Procedure
- Overview: PRINT Procedure ........................................................ 235
- Concepts: PRINT Procedure ......................................................... 236
- Syntax: PRINT Procedure .............................................................. 238
- Printing CAS Tables ................................................................. 241
- Error Processing in the PRINT Procedure Output ................ 258
- Examples: PRINT Procedure ....................................................... 258

### Chapter 13 • PRINTTO Procedure
- Overview: PRINTTO Procedure .................................................... 297
- Syntax: PRINTTO Procedure .......................................................... 297
- Setting Page Numbers Using SAS System Options .......... 302
- Restoring the Previous SAS Log or Output File Location ..... 302
- Examples: PRINTTO Procedure ..................................................... 303

### Chapter 14 • PRODUCT_STATUS Procedure
- Overview: PRODUCT_STATUS Procedure .................................. 311
- Syntax: PRODUCT_STATUS Procedure ...................................... 311
- Example: Results from PROC PRODUCT_STATUS ............. 312

### Chapter 15 • PWENCODE Procedure
- Overview: PWENCODE Procedure .............................................. 313
- Concepts: PWENCODE Procedure .............................................. 313
- Syntax: PWENCODE Procedure ..................................................... 314
Chapter 16 • SORT Procedure .................................................. 319
   Overview: SORT Procedure .............................................. 320
   Concepts: SORT Procedure .............................................. 321
   Syntax: SORT Procedure ................................................ 329
   In-Database Processing: PROC SORT .................................. 346
   Integrity Constraints: SORT Procedure ............................ 348
   Specifying the Host Sort Utility ..................................... 348
   Specifying the SORTSEQ= Option with a Host Sort Utility ... 349
   Results: SORT Procedure ............................................... 350
   Examples: SORT Procedure .............................................. 351

Chapter 17 • TRANSPOSE Procedure ....................................... 367
   Overview: TRANSPOSE Procedure ..................................... 367
   Syntax: TRANSPOSE Procedure ........................................ 371
   Results: TRANSPOSE Procedure ....................................... 378
   Examples: TRANSPOSE Procedure ..................................... 380

PART 2 Appendixes 395

Appendix 1 • Raw Data and DATA Steps for Base SAS Procedures .......... 397
   Overview of Raw Data and DATA Steps for Base SAS Procedures .... 398
   AddressPrint .............................................................. 398
   CARSURVEY ............................................................. 399
   CENSUS ................................................................. 400
   CHARITY ................................................................. 401
   CUSTOMER_RESPONSE .............................................. 403
   DJIA ................................................................. 406
   EDUCATION ........................................................... 407
   EMPDATA .............................................................. 407
   ENERGY ............................................................... 410
   EXP Library ............................................................ 412
   EXPREV .............................................................. 413
   GROC ................................................................. 414
   MATCH_11 ............................................................. 415
   PROCLIB.DELAY ...................................................... 416
   PROCLIB.EMP95 ...................................................... 417
   PROCLIB.EMP96 ...................................................... 418
   PROCLIB.INTERNAT .................................................. 419
   PROCLIB.LAKES ...................................................... 419
   PROCLIB.PAYROLL .................................................... 420
   PROCLIB.STAFF ...................................................... 423
   RADIO ............................................................... 426
   SALES ................................................................. 438

Appendix 2 • ICU License ....................................................... 439
   ICU License - ICU 1.8.1 and later .................................... 439
   Third-Party Software Licenses ........................................ 440

Recommended Reading ................................................................ 447
Index ................................................................................. 449
THIS DOCUMENTATION FOR AN EARLY ADOPTER PRODUCT IS A PRELIMINARY DRAFT AND IS PROVIDED BY SAS INSTITUTE INC. ("SAS") ON AN "AS IS" BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE. SAS does not warrant that this documentation is complete, accurate, or similar to that which may be released to the general public, or that any such documentation will be released. The company shall not be liable whatsoever for any damages arising out of the use of this documentation, including any direct, indirect, or consequential damages. SAS reserves the right to alter or abandon use of this documentation at any time.
Early Adopter Software
Part 1

Procedures

Chapter 1
APPEND Procedure .................................................. 3

Chapter 2
CONTENTS Procedure ............................................. 13

Chapter 3
COPY Procedure ..................................................... 35

Chapter 4
DATASETS Procedure ............................................. 43

Chapter 5
DELETE Procedure .................................................. 69

Chapter 6
EXPORT Procedure .................................................. 73

Chapter 7
FORMAT Procedure ................................................ 95

Chapter 8
HTTP Procedure ..................................................... 175

Chapter 9
IMPORT Procedure .................................................. 193

Chapter 10
JAVINFO Procedure ................................................. 217

Chapter 11
OPTIONS Procedure ................................................ 219

Chapter 12
PRINT Procedure ................................................... 235

Chapter 13
PRINTTO Procedure ................................................. 297
Chapter 14
   PRODUCT_STATUS Procedure ........................................... 311

Chapter 15
   PWENCODE Procedure ................................................... 313

Chapter 16
   SORT Procedure ............................................................ 319

Chapter 17
   TRANSPOSE Procedure .................................................. 367
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.
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:

```plaintext
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.

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC APPEND</td>
<td>Add rows from a SAS data set or a CAS table to the end of another SAS data set</td>
<td>Ex. 1</td>
</tr>
</tbody>
</table>

**PROC APPEND Statement**

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

**Syntax**

```sas
PROC APPEND BASE=<libref.>data-set
```
PROC APPEND Statement 7

<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:

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

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>SASCAS1.LASTMONTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
</tr>
<tr>
<td>Engine</td>
<td>CAS</td>
</tr>
<tr>
<td>Created</td>
<td>04/20/2010 13:30:55</td>
</tr>
<tr>
<td>Last Modified</td>
<td>04/20/2018 13:38:58</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
</tr>
<tr>
<td>Data Representation</td>
<td>SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRUNC, LINUX_IA64</td>
</tr>
<tr>
<td>Encoding</td>
<td>UTF-8 Unicode (UTF-8)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>address</td>
<td>Varch</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>city</td>
<td>Varch</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>month</td>
<td>Char</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>state</td>
<td>Char</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>store_id</td>
<td>Char</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>territories</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>zipcode</td>
<td>Char</td>
<td>5</td>
</tr>
</tbody>
</table>
The Monthly Data Set Contents

### The CONTENTS Procedure

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>SALESLIB MONTHLY</th>
<th>Observations</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>7</td>
</tr>
<tr>
<td>Engine</td>
<td>V9</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>03/31/2018 10:53:34</td>
<td>Observation Length</td>
<td>288</td>
</tr>
<tr>
<td>Last Modified</td>
<td>03/31/2018 10:53:34</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>utf-8 Unicode (UTF-8)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Engine/Host Dependent Information

<table>
<thead>
<tr>
<th>Data Set Page Size</th>
<th>05000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Data Set Pages</td>
<td>1</td>
</tr>
<tr>
<td>First Data Page</td>
<td>1</td>
</tr>
<tr>
<td>Max Obs per Page</td>
<td>227</td>
</tr>
<tr>
<td>Obs in First Data Page</td>
<td>12</td>
</tr>
<tr>
<td>Number of Data Set Repairs</td>
<td>0</td>
</tr>
<tr>
<td>Filename</td>
<td></td>
</tr>
<tr>
<td>Release Created</td>
<td>V.0300M3</td>
</tr>
<tr>
<td>Host Created</td>
<td>Linux</td>
</tr>
<tr>
<td>Inode Number</td>
<td>60980734</td>
</tr>
<tr>
<td>Access Permission</td>
<td>rw---r--</td>
</tr>
<tr>
<td>Owner Name</td>
<td>suhdm</td>
</tr>
<tr>
<td>File Size</td>
<td>126KB</td>
</tr>
<tr>
<td>File Size (bytes)</td>
<td>131072</td>
</tr>
</tbody>
</table>

### Alphabetical List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>address</td>
<td>Char</td>
<td>160</td>
</tr>
<tr>
<td>4</td>
<td>city</td>
<td>Char</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>month</td>
<td>Char</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>state</td>
<td>Char</td>
<td>2</td>
</tr>
</tbody>
</table>

Concatenated Data Set

### The CONTENTS Procedure

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>SALESLIB MONTHLY</th>
<th>Observations</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>7</td>
</tr>
<tr>
<td>Engine</td>
<td>V9</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>04/11/2018 13:58:42</td>
<td>Observation Length</td>
<td>200</td>
</tr>
<tr>
<td>Last Modified</td>
<td>04/11/2018 13:58:42</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>utf-8 Unicode (UTF-8)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Output 1.4  Total Sales

<table>
<thead>
<tr>
<th>store_id</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zipcode</th>
<th>totalsales</th>
</tr>
</thead>
<tbody>
<tr>
<td>842</td>
<td>318 S Barnes St</td>
<td>What Cheer IA</td>
<td>50283</td>
<td></td>
<td>$2,219,407</td>
</tr>
<tr>
<td>488</td>
<td>2346 Lindeman Lr</td>
<td>Nameless TX</td>
<td>78841</td>
<td></td>
<td>$2,101,480</td>
</tr>
<tr>
<td>814</td>
<td>110 Tuscarora Pike</td>
<td>Shanghai WV</td>
<td>25427</td>
<td></td>
<td>$2,190,375</td>
</tr>
</tbody>
</table>
Chapter 2

CONTENTS Procedure

Overview: CONTENTS Procedure ............................................. 13
Concepts: CONTENTS Procedure ............................................. 15
  Printing Variables ......................................................... 15
Syntax: CONTENTS Procedure ................................................ 15
  PROC CONTENTS Statement ............................................. 15
Results: CONTENTS Procedure ............................................. 20
  Output Tables ............................................................. 20
Examples: CONTENTS Procedure .......................................... 25
  Example 1: Describing a SAS Data Set .............................. 25
  Example 2: Using the DIRECTORY Option ......................... 29
  Example 3: Using the DIRECTORY and DETAILS Options ...... 31

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

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

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

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

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. 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>;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CONTENTS</td>
<td>List the contents of one or more CAS tables and print the directory of the caslib</td>
<td>Ex. 1, Ex. 2, Ex. 3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>To</th>
<th>CLONE or NOCLONE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFSIZE=</td>
<td>SAS data set to CAS table</td>
<td>NOCLONE</td>
<td>Uses setting of BUFSIZE= system option</td>
</tr>
<tr>
<td>CAS engine does not support.</td>
<td>CAS table to CAS table</td>
<td>CLONE</td>
<td>OVERRIDE=(BUFSIZE=v alue other than default)</td>
</tr>
<tr>
<td></td>
<td>CAS table to SAS data set</td>
<td>NOCLONE</td>
<td>Uses setting of BUFSIZE= system option</td>
</tr>
</tbody>
</table>

16 Chapter 2 • CONTENTS Procedure
<table>
<thead>
<tr>
<th>Attribute</th>
<th>To</th>
<th>CLONE or NOCLONE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESS=</td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Follows the CAS LIBNAME setting.</td>
</tr>
<tr>
<td></td>
<td>CAS table to SAS data set</td>
<td>CLONE</td>
<td>Compressed CAS table becomes a SAS data set CHAR unless the OVERRIDE= is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>The COMPRESS= system option or LIBNAME option value is used.</td>
</tr>
<tr>
<td></td>
<td>CAS table to CAS table</td>
<td>CLONE</td>
<td>Keeps the current setting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Set using the CAS LIBNAME setting for the OUT= libref.</td>
</tr>
<tr>
<td>REUSE=</td>
<td>SAS data set to CAS table</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAS table to SAS data set</td>
<td>CLONE</td>
<td>REUSE=NO unless OVERRIDE= or REUSE=YES system option is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Uses the REUSE= system option value.</td>
</tr>
<tr>
<td></td>
<td>CAS table to CAS table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POINTOBS=</td>
<td>SAS data set to CAS table</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAS table to SAS data set</td>
<td>CLONE</td>
<td>POINTOBS=NO unless OVERRIDE= is used.</td>
</tr>
<tr>
<td>Attribute</td>
<td>To</td>
<td>CLONE or NOCLONE</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>NOCLONE</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>CAS table to CAS table</td>
<td>OUTREP=</td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
</tr>
<tr>
<td>CAS table to SAS data set</td>
<td></td>
<td></td>
<td>NOCLONE</td>
</tr>
<tr>
<td>CAS table to CAS table</td>
<td>ENCODING=</td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
</tr>
<tr>
<td>CAS table to SAS data set</td>
<td></td>
<td></td>
<td>NOCLONE</td>
</tr>
<tr>
<td>CAS table to SAS data set</td>
<td></td>
<td></td>
<td>NOCLONE</td>
</tr>
<tr>
<td>CAS table to SAS data set</td>
<td></td>
<td></td>
<td>NOCLONE</td>
</tr>
</tbody>
</table>
**Attribute** | **To** | **CLONE or NOCLONE** | **Description**
--- | --- | --- | ---

| 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.
 DELOBBS  
  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 — Section 1

<table>
<thead>
<tr>
<th>Obs</th>
<th>LIBNAME</th>
<th>MEMNAME</th>
<th>MEMLABEL</th>
<th>TYPEMEM</th>
<th>NAME</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>VARNUM</th>
<th>LABEL</th>
<th>FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Cylinders</td>
<td>1</td>
<td>8</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Lineweight</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>EngineSize</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>Engine Size (L)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Horsepower</td>
<td>1</td>
<td>6</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Length</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>Length (IN)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>MPG_City</td>
<td>1</td>
<td>6</td>
<td>10</td>
<td>MPG (City)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>MPG_Highway</td>
<td>1</td>
<td>8</td>
<td>11</td>
<td>MPG (Highway)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>MSRP</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>DOLLAR</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Make</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Model</td>
<td>8</td>
<td>40</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Origin</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Type</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Weight</td>
<td>1</td>
<td>6</td>
<td>12</td>
<td>Weight (LBS)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Wheelbase</td>
<td>1</td>
<td>8</td>
<td>13</td>
<td>Wheelbase (IN)</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>MYCAS</td>
<td>CARS</td>
<td></td>
<td></td>
<td>Invoice</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>DOLLAR</td>
<td></td>
</tr>
<tr>
<td>FORMATL</td>
<td>FORMATD</td>
<td>INFORMAT</td>
<td>INFORMDL</td>
<td>INFORMMD</td>
<td>JUST</td>
<td>NPOS</td>
<td>NOBS</td>
<td>ENGINE</td>
<td>CRDATE</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>--------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>120</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46</td>
<td>426</td>
<td>CAS3</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>72</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>129</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>80</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>88</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>96</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>64</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>104</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>112</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>56</td>
<td>428</td>
<td>CAS</td>
<td>20APR16:12:03:47</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MODATE</th>
<th>DELCBS</th>
<th>IDX3AGE</th>
<th>MEMTYPE</th>
<th>IDXCOUNT</th>
<th>PROTECT</th>
<th>FLAGS</th>
<th>COMPRESS</th>
<th>REUSE</th>
<th>SORTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>20APR16:16:07:68</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:58</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:58</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:56</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:56</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:56</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:56</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:56</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:56</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:56</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
<tr>
<td>20APR16:16:07:56</td>
<td>0</td>
<td>NONE</td>
<td>DATA</td>
<td>0</td>
<td>---</td>
<td>---</td>
<td>NO</td>
<td>NO</td>
<td>.</td>
</tr>
</tbody>
</table>
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**

```sas
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;
quit;

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

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>MYCAS.CARS</th>
<th>Observations</th>
<th>420</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>15</td>
</tr>
<tr>
<td>Engine</td>
<td>CAS</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>04/19/2010 11:44:24</td>
<td>Observation Length</td>
<td>100</td>
</tr>
<tr>
<td>Last Modified</td>
<td>04/19/2010 11:44:24</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>utf-8 Unencoded (UTF-8)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Alphabetic List of Variables and Attributes**

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

**Alphabetic List of Variables and Attributes**

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Cylinders</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DriveTrain</td>
<td>Char</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>EngineSize</td>
<td>Num</td>
<td>8</td>
<td>Engine Size (L)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Horsepower</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Invoice</td>
<td>Num</td>
<td>8</td>
<td>DOLLARS.</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Length</td>
<td>Num</td>
<td>8</td>
<td>Length (IN)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>MPG_City</td>
<td>Num</td>
<td>8</td>
<td>MPG (City)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MPG_Highway</td>
<td>Num</td>
<td>8</td>
<td>MPG (Highway)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MSRP</td>
<td>Num</td>
<td>2</td>
<td>DOLLARS.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Make</td>
<td>Char</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Model</td>
<td>Char</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Origin</td>
<td>Char</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Type</td>
<td>Char</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Weight</td>
<td>Num</td>
<td>8</td>
<td>Weight (LBS)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Wheelbase</td>
<td>Num</td>
<td>8</td>
<td>Wheelbase (IN)</td>
<td></td>
</tr>
</tbody>
</table>
### Contents of the CarsOut Table

#### The CONTENTS Procedure

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>MYCAS.CARSOUT</th>
<th>Observations</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>41</td>
</tr>
<tr>
<td>Engine</td>
<td>CAS</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>03/30/2016 10:34:00</td>
<td>Observation Length</td>
<td>928</td>
</tr>
<tr>
<td>Last Modified</td>
<td>03/30/2016 10:34:20</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>UTF-8 Unicode (UTF-8)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>CHARSET</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Host Character Set</td>
</tr>
<tr>
<td>33</td>
<td>COLLATE</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Collating Sequence</td>
</tr>
<tr>
<td>28</td>
<td>COMPRESS</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Compression Routine</td>
</tr>
<tr>
<td>29</td>
<td>CREATE</td>
<td>Num</td>
<td>8</td>
<td>DATE</td>
<td>Grease Data</td>
</tr>
<tr>
<td>22</td>
<td>DELOBB</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Deleted Observations In Data Set</td>
</tr>
<tr>
<td>36</td>
<td>ENCRYPT</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Encryption Routine</td>
</tr>
<tr>
<td>19</td>
<td>ENGINE</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Engine Name</td>
</tr>
<tr>
<td>27</td>
<td>FLAGS</td>
<td>Char</td>
<td>3</td>
<td></td>
<td>Update Flags (Protect, Contribute, Acid)</td>
</tr>
<tr>
<td>10</td>
<td>FORMAT</td>
<td>Char</td>
<td>32</td>
<td></td>
<td>Variable Format</td>
</tr>
<tr>
<td>12</td>
<td>FORMATTD</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Number of Format Decimals</td>
</tr>
<tr>
<td>11</td>
<td>FORMATL</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Format Length</td>
</tr>
<tr>
<td>38</td>
<td>GENMAX</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Maximum Number of Generations</td>
</tr>
<tr>
<td>40</td>
<td>GENVEXT</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Next Generation Number</td>
</tr>
<tr>
<td>39</td>
<td>GENSEM</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Generation Number</td>
</tr>
<tr>
<td>25</td>
<td>INDEXCOUNT</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Number of Indexes For Data Set</td>
</tr>
<tr>
<td>23</td>
<td>INDEXNAME</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Use of Variable In Indexes</td>
</tr>
<tr>
<td>13</td>
<td>INFORMAT</td>
<td>Char</td>
<td>32</td>
<td></td>
<td>Variable Informat</td>
</tr>
<tr>
<td>15</td>
<td>INFORMID</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Number of Informat Decimals</td>
</tr>
<tr>
<td>14</td>
<td>INFORMIL</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Informat Length</td>
</tr>
<tr>
<td>16</td>
<td>JUST</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Justification</td>
</tr>
<tr>
<td>9</td>
<td>LABEL</td>
<td>Char</td>
<td>255</td>
<td></td>
<td>Variable Label</td>
</tr>
<tr>
<td>7</td>
<td>LENGTH</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Variable Length</td>
</tr>
<tr>
<td>1</td>
<td>LIBNAME</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Library Name</td>
</tr>
<tr>
<td>3</td>
<td>MEMLABEL</td>
<td>Char</td>
<td>255</td>
<td></td>
<td>Data Set Label</td>
</tr>
<tr>
<td>2</td>
<td>MEMNAME</td>
<td>Char</td>
<td>32</td>
<td></td>
<td>Library Member Name</td>
</tr>
<tr>
<td>24</td>
<td>MEMTYPE</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Library Member Type</td>
</tr>
<tr>
<td>21</td>
<td>MODDATE</td>
<td>Num</td>
<td>8</td>
<td>DATETIME10</td>
<td>Last Modified Date</td>
</tr>
<tr>
<td>5</td>
<td>NAME</td>
<td>Char</td>
<td>32</td>
<td></td>
<td>Variable Name</td>
</tr>
<tr>
<td>18</td>
<td>NOOGS</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Observations In Data Set</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libref</td>
</tr>
<tr>
<td>Engine</td>
</tr>
<tr>
<td>Physical Name</td>
</tr>
<tr>
<td>Server Session UUID</td>
</tr>
<tr>
<td>Server Session Name</td>
</tr>
<tr>
<td>Server Host</td>
</tr>
<tr>
<td>Server Session Port</td>
</tr>
<tr>
<td>Server Session CASLIB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Member Type</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Last Modified</th>
<th>Data Encoding</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CARS</td>
<td>DATA</td>
<td>428</td>
<td>16</td>
<td>04/16/2016 11:34:00</td>
<td>utf-8</td>
<td>Session</td>
</tr>
<tr>
<td>2</td>
<td>LASTMONTH</td>
<td>DATA</td>
<td>12</td>
<td>7</td>
<td>04/16/2016 11:33:32</td>
<td>utf-8</td>
<td>Session</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libref</td>
</tr>
<tr>
<td>Engine</td>
</tr>
<tr>
<td>Physical Name</td>
</tr>
<tr>
<td>Server Session UUID</td>
</tr>
<tr>
<td>Server Session Name</td>
</tr>
<tr>
<td>Server Host</td>
</tr>
<tr>
<td>Server Session Port</td>
</tr>
<tr>
<td>Server Session CASLIB</td>
</tr>
</tbody>
</table>

Using the DIRECTORY and DETAILS Options

The CONTENTS Procedure

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>MYCAS.CARS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observations</td>
<td>428</td>
</tr>
<tr>
<td>Member Type</td>
<td>DATA</td>
</tr>
<tr>
<td>Variables</td>
<td>15</td>
</tr>
<tr>
<td>Engine</td>
<td>CAS</td>
</tr>
<tr>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>03/01/2016 15:31:25</td>
</tr>
<tr>
<td>Observation Length</td>
<td>100</td>
</tr>
<tr>
<td>Last Modified</td>
<td>03/01/2016 15:31:25</td>
</tr>
<tr>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
</tr>
<tr>
<td>Label</td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>SOLARIS_X96_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64</td>
</tr>
<tr>
<td>Encoding</td>
<td>utf-8 Unicode (UTF-8)</td>
</tr>
</tbody>
</table>

Alphabetic List of Variables and Attributes

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

Copying a CAS Table to Another CAS Table

Copying an Entire Library

Copying Selected Tables

Copying Compressed Tables

Syntax: COPY Procedure
PROC COPY Statement

Using the COPY Procedure
Compressing Output Tables

Example: Copy a SAS Data Set to a CAS Table
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:

```sas
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:

```sas
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:

```sas
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 …> ) >;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC COPY</td>
<td>Copy one or more tables in a library</td>
<td>Ex. 1</td>
</tr>
</tbody>
</table>

---

**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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>To</th>
<th>CLONE or NOCLONE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFSIZE=</td>
<td>SAS data set to CAS table</td>
<td></td>
<td>CAS engine does not support.</td>
</tr>
<tr>
<td>COMPRESS=</td>
<td>SAS data set to CAS table</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>CAS engine does not support.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPRESS=BINARY</td>
<td>CAS table to SAS data set</td>
<td>CLONE</td>
<td>OVERRIDE=(BUFSIZE=value other than default)</td>
</tr>
<tr>
<td>COMPRESS=NO</td>
<td>CAS table to SAS data set</td>
<td>NOCLONE</td>
<td>Uses setting of BUFSIZE= system option</td>
</tr>
<tr>
<td>COMPRESS=YES</td>
<td>CAS table to CAS table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPRESS=BINARY</td>
<td>CAS table to SAS data set</td>
<td>CLONE</td>
<td></td>
</tr>
<tr>
<td>COMPRESS=NO</td>
<td>CAS table to CAS table</td>
<td>NOCLONE</td>
<td></td>
</tr>
<tr>
<td>COMPRESS=YES</td>
<td>CAS table to CAS table</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

38 Chapter 3 • COPY Procedure
<table>
<thead>
<tr>
<th>Attribute</th>
<th>To</th>
<th>CLONE or NOCLONE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CAS table to CAS table</td>
<td>CLONE</td>
<td>Keeps the current setting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Set using the CAS LIBNAME setting for the OUT= libref.</td>
</tr>
<tr>
<td>REUSE=</td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
<td>REUSE=NO unless the OVERRIDE= or REUSE=YES system options are used.</td>
</tr>
<tr>
<td>CAS engine does not</td>
<td></td>
<td>NOCLONE</td>
<td>Uses the REUSE= system option value.</td>
</tr>
<tr>
<td>POINTOBS=</td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
<td>POINTOBS=NO unless OVERRIDE= is used.</td>
</tr>
<tr>
<td>CAS engine does not</td>
<td></td>
<td>NOCLONE</td>
<td>POINTOBS=NO, if the CAS table is compressed and the LIBNAME statement has</td>
</tr>
<tr>
<td></td>
<td>CAS table to SAS data set</td>
<td></td>
<td>POINTOBS=NO.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>POINTOBS=YES, if the CAS table is compressed and the LIBNAME option is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>missing.</td>
</tr>
<tr>
<td>OUTREP=</td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
<td>Converts to LINUX_86_64 if needed. (A warning is sent to the log if the</td>
</tr>
<tr>
<td>CAS engine does not</td>
<td></td>
<td>NOCLONE</td>
<td>OVERRIDE= option is used.)</td>
</tr>
<tr>
<td></td>
<td>CAS table to CAS table</td>
<td></td>
<td>Converts to LINUX_86_64 if needed.</td>
</tr>
<tr>
<td>Attribute</td>
<td>To</td>
<td>CLONE or NOCLONE</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CAS table to SAS data set</td>
<td></td>
<td>CLONE</td>
<td>Keeps data representation. (A warning is sent to the log if the OVERRIDE= option is used.)</td>
</tr>
<tr>
<td>CAS table to CAS table</td>
<td></td>
<td>NOCLONE</td>
<td>Converts to UTF-8 if needed.</td>
</tr>
<tr>
<td>SAS data set to CAS table</td>
<td></td>
<td>CLONE</td>
<td>Converts to UTF-8 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Converts to UTF-8 if needed.</td>
</tr>
<tr>
<td>CAS table to SAS data set</td>
<td></td>
<td>CLONE</td>
<td>Keeps the UTF-8 encoding unless OVERRIDE= is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Keeps the UTF-8 encoding unless OUTENCODING= is used in the output SAS data set LIBNAME is used.</td>
</tr>
</tbody>
</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

```sas
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.

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

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

```sas
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;
```
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. For more information about the SAS 9.4 DATASETS procedure, see Base SAS 9.4 Procedures Guide on support.sas.com/documentation.

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)>;
**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:

```plaintext
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:

```plaintext
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.
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.

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

Output 4.1  Mycas Caslib and Mycas.Cars Table

<table>
<thead>
<tr>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libref</td>
</tr>
<tr>
<td>Engine</td>
</tr>
<tr>
<td>Physical Name</td>
</tr>
<tr>
<td>Server Session UUID</td>
</tr>
<tr>
<td>Server Session Name</td>
</tr>
<tr>
<td>Server Host</td>
</tr>
<tr>
<td>Server Session Port</td>
</tr>
<tr>
<td>Server Session CASLIB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Member Type</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Last Modified</th>
<th>Data Encoding</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CARS</td>
<td>DATA</td>
<td>420</td>
<td>15</td>
<td>04/18/2018 11:44:24</td>
<td>utf-8</td>
<td>Session</td>
</tr>
<tr>
<td>2</td>
<td>CARSCUT</td>
<td>DATA</td>
<td>16</td>
<td>41</td>
<td>04/18/2018 11:44:24</td>
<td>utf-8</td>
<td>Session</td>
</tr>
<tr>
<td>3</td>
<td>LASTMONTH</td>
<td>DATA</td>
<td>12</td>
<td>7</td>
<td>04/18/2018 11:33:32</td>
<td>utf-8</td>
<td>Session</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>To</th>
<th>CLONE or NOCLONE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFSIZE=</td>
<td>SAS data set to CAS table</td>
<td></td>
<td>CAS engine does not support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPRESS=</td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
<td>OVERRIDE=(BUFSIZE=value other than default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Uses setting of BUFSIZE= system option</td>
</tr>
<tr>
<td></td>
<td>CAS table to SAS data set</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLONE</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Follows the CAS LIBNAME setting.</td>
</tr>
<tr>
<td></td>
<td>CAS table to CAS table</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLONE</td>
<td>Compressed CAS table becomes a SAS data set CHAR value variable unless the OVERRIDE= is used.</td>
</tr>
<tr>
<td>Attribute</td>
<td>To</td>
<td>CLONE or NOCLONE</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>The COMPRESS= system option or LIBNAME option value is used.</td>
</tr>
<tr>
<td></td>
<td>CAS table to CAS table</td>
<td>CLONE</td>
<td>Keeps the current setting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Set using the CAS LIBNAME setting for the OUT= libref.</td>
</tr>
<tr>
<td>REUSE=</td>
<td>SAS data set to CAS table</td>
<td></td>
<td>REUSE=NO unless the OVERRIDE= or REUSE=YES system options are used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Uses the REUSE= system option value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POINTOBS=</td>
<td>SAS data set to CAS table</td>
<td></td>
<td>POINTOBS=NO unless OVERRIDE= is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTREP=</td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
<td>Converts to LINUX_86_64 if needed. (A warning is sent to the log if the OVERRIDE= option is used.)</td>
</tr>
<tr>
<td>Attribute</td>
<td>To</td>
<td>CLONE or NOCLONE</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOCLONE</td>
<td>Converts to LINUX_86_64 if needed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CAS table to SAS data set</td>
<td>CLONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS data set to CAS table</td>
<td>CLONE</td>
</tr>
<tr>
<td>ENCODING=</td>
<td></td>
<td>NOCLONE</td>
<td>Converts to UTF-8 if needed.</td>
</tr>
<tr>
<td>CAS engine supports UTF-8 only.</td>
<td></td>
<td>CLONE</td>
<td>Keeps the UTF-8 encoding unless OVERRIDE= is used.</td>
</tr>
<tr>
<td>Changing encoding is not implemented for the CAS engine.</td>
<td></td>
<td>NOCLONE</td>
<td>Keeps the UTF-8 encoding unless OUTENCODING= is used in the output data set LIBNAME is used.</td>
</tr>
<tr>
<td>IN=libref-2</td>
<td></td>
<td>CLONE</td>
<td>Keeps the UTF-8 encoding unless OUTENCODING= is used in the output data set LIBNAME is used.</td>
</tr>
<tr>
<td>names the library containing CAS tables to copy.</td>
<td>INLIB= and INDD=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alias</td>
<td>MEMTYPE=(member-type(s))</td>
<td>restricts processing to one or more member types. Member types available for the CAS engine are DATA and ALL.</td>
<td>MT=, MTYPE=</td>
</tr>
<tr>
<td>Default</td>
<td>Interaction</td>
<td>To copy only selected CAS tables, use the SELECT or EXCLUDE statements.</td>
<td></td>
</tr>
</tbody>
</table>
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 35.

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:
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:

```sas
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:

```sas
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:

```sas
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:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1–xn</td>
<td>Specifies tables X1 through Xn. The numbers must be consecutive.</td>
</tr>
<tr>
<td>x:</td>
<td>Specifies all tables that begin with the letter X.</td>
</tr>
</tbody>
</table>

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.

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:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1–xn</td>
<td>Specifies tables X1 through Xn. The numbers</td>
</tr>
<tr>
<td></td>
<td>must be consecutive.</td>
</tr>
<tr>
<td>x:</td>
<td>Specifies all tables that begin with the</td>
</tr>
<tr>
<td></td>
<td>letter X.</td>
</tr>
</tbody>
</table>

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, Entries or 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.
Using the DIRECTORY Option

The CONTENTS Procedure

| Data Set Name | MYCAS.CARS | Observations | 420 |
| Member Type   | DATA       | Variables    | 16  |
| Engine        | CAS        | Indexes      | 0   |
| Created       | 02/20/2018 17:16:00 | Observation Length | 160 |
| Last Modified | 02/20/2018 17:16:00 | 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) |

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Cylinders</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DriveTrain</td>
<td>Char</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>EngineSize</td>
<td>Num</td>
<td>8</td>
<td>Engine Size</td>
<td>Engine Size (L)</td>
</tr>
<tr>
<td>10</td>
<td>Horsepower</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Invoice</td>
<td>Num</td>
<td>8</td>
<td>DOLLAR3.</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Length</td>
<td>Num</td>
<td>8</td>
<td>Length (IN)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>MPG_City</td>
<td>Num</td>
<td>8</td>
<td>MPG (City)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MPG_Highway</td>
<td>Num</td>
<td>8</td>
<td>MPG (Highway)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MSRP</td>
<td>Num</td>
<td>8</td>
<td>DOLLAR3.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Make</td>
<td>Char</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Model</td>
<td>Char</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Origin</td>
<td>Char</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Type</td>
<td>Char</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Weight</td>
<td>Num</td>
<td>8</td>
<td>Weight (LBS)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Wheelbase</td>
<td>Num</td>
<td>8</td>
<td>Wheelbase (IN)</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>ODS Table</th>
<th>Description</th>
<th>Generates Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
<td>General library information</td>
<td>Unless you specify the NOLIST option</td>
</tr>
<tr>
<td>Members</td>
<td>Library member information</td>
<td>Unless you specify the NOLIST option</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>ODS Table</th>
<th>Description</th>
<th>Generates Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Data set attributes</td>
<td>Unless you specify the SHORT option</td>
</tr>
</tbody>
</table>
### ODS Table

<table>
<thead>
<tr>
<th>ODS Table</th>
<th>Description</th>
<th>Generates Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
<td>General library information</td>
<td>If you specify DATA=&lt;<em>libref:</em> ALL_ or the DIRECTORY option</td>
</tr>
<tr>
<td>Members</td>
<td>Library member information</td>
<td>If you specify DATA=&lt;<em>libref:</em> ALL_ or the DIRECTORY option</td>
</tr>
<tr>
<td>Position</td>
<td>A detailed listing of variables by logical position in the table</td>
<td>If you specify the VARNUM option and you do not specify the SHORT option</td>
</tr>
<tr>
<td>PositionShort</td>
<td>A concise listing of variables by logical position in the table</td>
<td>If you specify the VARNUM option and the SHORT option</td>
</tr>
<tr>
<td>Variables</td>
<td>A detailed listing of variables in alphabetical order</td>
<td>Unless you specify the SHORT option</td>
</tr>
<tr>
<td>VariablesShort</td>
<td>A concise listing of variables in alphabetical order</td>
<td>If you specify the SHORT option</td>
</tr>
</tbody>
</table>

### 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.
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

  ```plaintext
  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

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

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

Overview: Export Procedure

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.

---

**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:

```sas
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.

---

**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;**

---

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

---

**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
79

“Example 2: Exporting a Subset of Observations to a CSV File” on page
84

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 79

“Example 2: Exporting a Subset of Observations to a CSV File” on page 84

**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 6.1  DBMS Identifiers Supported in SAS**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Output Data Source</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV</td>
<td>Delimited file (comma-separated values)</td>
<td>.csv</td>
</tr>
<tr>
<td>DLM</td>
<td>Delimited file (default delimiter is a blank)</td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>JMP files, Version 7 or later format</td>
<td>.jmp</td>
</tr>
<tr>
<td>TAB</td>
<td>Delimited file (tab-delimited values)</td>
<td>.txt</td>
</tr>
</tbody>
</table>
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 79

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 84

(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 84

---

**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 79

**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 87
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 6.1  PROC PRINT of SASHelp.Class

The SAS System

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

Program

```sas
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.

```sas
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.

```sas
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 6.1  SAS Log of Creating a Delimited File**

```sas
options nonotes nosources nosyntaxcheck;
proc print data=sashelp.class;
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

proc export data=sashelp.class
   outfile="/userid/pathname/class.txt"
   dbms=tab replace;
   putnames=yes;
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.

[Example code]
```
do;
    put
    "Name"
    '09'x
    "Sex"
    '09'x
    "Age"
    '09'x
    "Height"
    '09'x
    "Weight"
    ;
    end;
set SASHELP.CLASS end=EFIEOD;
format Name $8. ;
format Sex $1. ;
format Age best12. ;
format Height best12. ;
format Weight best12. ;
do;
    EFIOUT + 1;
    put Name $ @;
    put Sex $ @;
    put Age @;
    put Height @;
    put Weight ;
    end;
if _ERROR_ then call symputx('_EFIERR_','.);
/* set ERROR detection macro variable */
if EFIEOD then call symputx('_EFIREC_',EFIOUT);
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=29Mar2016:08:45:33

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: PROCEDURE EXPORT used (Total process time):
real time 0.09 seconds
cpu time 0.04 seconds

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

NOTE: "/userid/pathname/class.txt" file was successfully created.

NOTE: PROCEDURE PRINT used (Total process time):
real time 0.02 seconds
cpu time 0.01 seconds

PROC PRINT;
RUN;
PROC EXPORT DATA=sashelp.class
  OUTFILE="/userid/pathname/class.txt"
  DBMS=TAB REPLACE;
  PUTNAMES=NO;
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.

/******************************************************************************
 *   PRODUCT:   SAS
 *   VERSION:   V.03.00
 *   CREATOR:   External File Interface
 *   DATE:      29MAR16
 *   DESC:      Generated SAS Datastep Code
 *   TEMPLATE SOURCE:  (None Specified.)
 ******************************************************************************/
data _null_
   %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
   %let _EFIREC_ = 0; /* clear export record count macro variable */
   file '/userid/pathname/class.txt' delimiter='09'x DSD DROPOVER lrecl=32767;
   set  SASHELP.CLASS   end=EFIEOD;
   format Name $8. ;
   format Sex $1. ;
   format Age best12. ;
   format Weight best12. ;
   do;
   EFIOUT + 1;
   put Name $ @;
   put Sex $ @;
   put Age @;
   put Height @;
   put Weight ;
   end;
   if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
   if EFIEOD then call symputx('_EFIREC_','EFIOUT');
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=29Mar2016:08:45:33

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 6.2 External File

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>M</td>
<td>14</td>
<td>69</td>
<td>112.5</td>
</tr>
<tr>
<td>Alice</td>
<td>F</td>
<td>13</td>
<td>56.5</td>
<td>84</td>
</tr>
<tr>
<td>Barbara</td>
<td>F</td>
<td>13</td>
<td>65.3</td>
<td>98</td>
</tr>
<tr>
<td>Carol</td>
<td>F</td>
<td>14</td>
<td>62.8</td>
<td>102.5</td>
</tr>
<tr>
<td>Henry</td>
<td>M</td>
<td>14</td>
<td>63.5</td>
<td>102.5</td>
</tr>
<tr>
<td>James</td>
<td>M</td>
<td>12</td>
<td>64.5</td>
<td>369</td>
</tr>
<tr>
<td>Jane</td>
<td>F</td>
<td>12</td>
<td>59.8</td>
<td>84.5</td>
</tr>
<tr>
<td>Jeffrey</td>
<td>M</td>
<td>13</td>
<td>65.2</td>
<td>5684</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>12</td>
<td>65</td>
<td>99.5</td>
</tr>
<tr>
<td>Joyce</td>
<td>F</td>
<td>11</td>
<td>51.3</td>
<td>50.5</td>
</tr>
<tr>
<td>Judy</td>
<td>F</td>
<td>14</td>
<td>65</td>
<td>3690</td>
</tr>
<tr>
<td>Louise</td>
<td>F</td>
<td>12</td>
<td>56.5</td>
<td>3677</td>
</tr>
<tr>
<td>Mary</td>
<td>F</td>
<td>15</td>
<td>66.5</td>
<td>5612</td>
</tr>
<tr>
<td>Philip</td>
<td>M</td>
<td>16</td>
<td>72</td>
<td>150</td>
</tr>
<tr>
<td>Robert</td>
<td>M</td>
<td>12</td>
<td>66.4</td>
<td>8128</td>
</tr>
<tr>
<td>Ronald</td>
<td>M</td>
<td>15</td>
<td>67</td>
<td>6133</td>
</tr>
<tr>
<td>Thomas</td>
<td>M</td>
<td>11</td>
<td>57.5</td>
<td>85</td>
</tr>
<tr>
<td>William</td>
<td>M</td>
<td>15</td>
<td>66.5</td>
<td>5612</td>
</tr>
</tbody>
</table>

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

```sas
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.

```sas
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.
OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;

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

/**********************************************************************
*   PRODUCT:   SAS
*   VERSION:   V.03.00
*   CREATOR:   External File Interface
*   DATE:      28MAR16
*   DESC:      Generated SAS Datastep Code
*   TEMPLATE SOURCE:  (None Specified.)
***********************************************************************/
data _null_;
%let _EFIERR_ = 0; /* set the ERROR detection macro variable */
%let _EFIREC_ = 0;     /* clear export record count macro variable */
file '/pathname/Femalelist.csv' delimiter=',' DSD DROPOVER lrecl=32767;
if _n_ = 1 then        /* write column names or labels */
do;
    put
    "Name"
    ',
    "Sex"
    ',
    "Age"
    ',
    "Height"
    ',
    "Weight";
end;
set  SASHELP.CLASS(where=(sex='F'))   end=EFIEOD;
format Name $8. ;
format Sex $1. ;
format Age best12. ;
format Height best12. ;
format Weight best12. ;
do;
EFIGOUT + 1;
put Name $ @;
put Sex $ @;
put Age @;
put Height @;
put Weight ;
end;
if _ERROR_ then call symputx('_EFIERR_','1'); /* set ERROR detection macro variable */
if EFIEOD then call symputx('_EFIREC_','EFIGOUT');
run;
NOTE: The file '/pathname/Femalelist.csv' is:
Filename=/pathname/Femalelist.csv,
Owner Name=userid, Group Name=unix_pubs,
Access Permission=rwx------,
Last Modified=28Mar2016:14:16:19

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

OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;

Output Examples

The EXPORT procedure produces this external CSV file:

Output 6.3 CSV File

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>F</td>
<td>13</td>
<td>56.5</td>
<td>84</td>
</tr>
<tr>
<td>Barbara</td>
<td>F</td>
<td>13</td>
<td>65.3</td>
<td>98</td>
</tr>
<tr>
<td>Carol</td>
<td>F</td>
<td>14</td>
<td>62.3</td>
<td>102.5</td>
</tr>
<tr>
<td>Jane</td>
<td>F</td>
<td>12</td>
<td>59.8</td>
<td>84.5</td>
</tr>
<tr>
<td>Janet</td>
<td>F</td>
<td>15</td>
<td>62.5</td>
<td>112.5</td>
</tr>
<tr>
<td>Joyce</td>
<td>F</td>
<td>11</td>
<td>51.3</td>
<td>50.5</td>
</tr>
<tr>
<td>Judy</td>
<td>F</td>
<td>14</td>
<td>64.3</td>
<td>90</td>
</tr>
<tr>
<td>Louise</td>
<td>F</td>
<td>12</td>
<td>56.3</td>
<td>77</td>
</tr>
<tr>
<td>Mary</td>
<td>F</td>
<td>15</td>
<td>66.5</td>
<td>112</td>
</tr>
</tbody>
</table>

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 6.4  PROC PRINT of WORK.INVOICE**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Invoice_ID</th>
<th>Billed_To</th>
<th>Amt_Billed</th>
<th>Country</th>
<th>Amt_in_US</th>
<th>Billed_By</th>
<th>Billed_On</th>
<th>Paid_On</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11270</td>
<td>30045213</td>
<td>373600640</td>
<td>Brazil</td>
<td>387543860</td>
<td>23915</td>
<td>05OCT2015</td>
<td>18OCT2015</td>
</tr>
<tr>
<td>2</td>
<td>11271</td>
<td>18543499</td>
<td>11083356</td>
<td>USA</td>
<td>11933838</td>
<td>457232</td>
<td>05OCT2015</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11273</td>
<td>19783462</td>
<td>252148.5</td>
<td>USA</td>
<td>252148.5</td>
<td>239165</td>
<td>09OCT2015</td>
<td>11NOV2015</td>
</tr>
<tr>
<td>4</td>
<td>11276</td>
<td>14324742</td>
<td>1924460</td>
<td>USA</td>
<td>1924460</td>
<td>135673</td>
<td>05OCT2015</td>
<td>20OCT2015</td>
</tr>
<tr>
<td>5</td>
<td>11278</td>
<td>14966029</td>
<td>1409625</td>
<td>USA</td>
<td>1409625</td>
<td>239165</td>
<td>09OCT2015</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11280</td>
<td>30045213</td>
<td>8738600640</td>
<td>Brazil</td>
<td>38858438662</td>
<td>423286</td>
<td>09OCT2015</td>
<td>20OCT2015</td>
</tr>
<tr>
<td>7</td>
<td>11282</td>
<td>19783482</td>
<td>252148.5</td>
<td>USA</td>
<td>252148.5</td>
<td>457232</td>
<td>07OCT2015</td>
<td>28OCT2015</td>
</tr>
<tr>
<td>8</td>
<td>11285</td>
<td>34863919</td>
<td>2234301.3</td>
<td>Argentina</td>
<td>772847.05</td>
<td>239165</td>
<td>10OCT2015</td>
<td>30NOV2015</td>
</tr>
<tr>
<td>9</td>
<td>11286</td>
<td>43459747</td>
<td>14836504.06</td>
<td>Australia</td>
<td>1135652.09</td>
<td>423286</td>
<td>10OCT2015</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11287</td>
<td>15432147</td>
<td>252148.5</td>
<td>USA</td>
<td>252148.5</td>
<td>457232</td>
<td>11OCT2015</td>
<td>04NOV2015</td>
</tr>
<tr>
<td>11</td>
<td>12051</td>
<td>30045213</td>
<td>8738600640</td>
<td>Brazil</td>
<td>38858438662</td>
<td>457232</td>
<td>02NOV2015</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12052</td>
<td>18543489</td>
<td>11053836</td>
<td>USA</td>
<td>11053836</td>
<td>239165</td>
<td>17NOV2015</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>12263</td>
<td>19783482</td>
<td>252148.5</td>
<td>USA</td>
<td>252148.5</td>
<td>423286</td>
<td>05DE2015</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>12468</td>
<td>14966029</td>
<td>1409625</td>
<td>USA</td>
<td>1409625</td>
<td>135673</td>
<td>24DE2015</td>
<td>02JAN2016</td>
</tr>
<tr>
<td>15</td>
<td>12471</td>
<td>30045213</td>
<td>8738600640</td>
<td>Brazil</td>
<td>387543860</td>
<td>457232</td>
<td>27DE2015</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>12475</td>
<td>38783919</td>
<td>2234301.3</td>
<td>Argentina</td>
<td>772847.05</td>
<td>135673</td>
<td>24DE2015</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>12478</td>
<td>15432147</td>
<td>252148.5</td>
<td>USA</td>
<td>252148.5</td>
<td>423286</td>
<td>24DE2015</td>
<td>02JAN2016</td>
</tr>
</tbody>
</table>

Program

```sas
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.

```sas
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.

```sas
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 6.3  Exporting with the PUTNAMES Statement

OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;

NOTE: Libref MYCAS was successfully assigned as follows:
    Engine:        CAS
    Physical Name: 19801cc2-65c1-4742-acfb-74ee86221874

PROC PRINT DATA=WORK.INVOICE;
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

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

/**********************************************************************
*   PRODUCT:   SAS
*   VERSION:   V.03.00
*   CREATOR:   External File Interface
*   DATE:      29MAR16
*   DESC:      Generated SAS Datastep Code
*   TEMPLATE SOURCE:  (None Specified.)
***********************************************************************/
data _null_
    &let _EFIERR_ = 0; /* set the ERROR detection macro variable */
    &let _EFIREC_ = 0; /* clear export record count macro variable */
    file '/userid/pathname/MyDocuments/invoice_names.txt' delimiter='09'x DSD DROPOVER
    lrecl=32767;
    if _n_ = 1 then /* write column names or labels */
    do;
    put
        "Invoice_ID"
        '09'x
        "Billed_To"
        '09'x
        "Amount_Billed_In_Local_Currency"
        '09'x
        "Country"
        '09'x
        "Amount_Billed_In_US_Dollars"
        '09'x
        "Billed_By"
        '09'x
        "Billed_On"
        '09'x
        "Paid_On"
    ;
end;
Example 3: Exporting to a Tab Delimited File with the PUTNAMES= Statement

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=29Mar2016:16:34:34

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
PROC EXPORT DATA=WORK.INVOICE
   OUTFILE="/userid/pathname/MyDocuments/invoice_data_1st.txt"
   DBMS=TAB REPLACE;
   PUTNAMES=NO;
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.

/******************************************************************************
*   PRODUCT:   SAS
*   VERSION:   V.03.00
*   CREATOR:   External File Interface
*   DATE:      29MAR16
*   DESC:      Generated SAS Datastep Code
*   TEMPLATE SOURCE:  (None Specified.)
*********************************************************************************/
data _null_; /* PRODUCT: SAS
* VERSION: V.03.00
* CREATOR: External File Interface
* DATE: 29MAR16
* DESC: Generated SAS Datastep Code
* TEMPLATE SOURCE: (None Specified.)
******************************************************************************/
   set WORK.INVOICE end=EFIEOD;
   format Invoice_ID best12. ;
   format Billed_To best12. ;
   format Amount_Billed_in_Local_Currency best12. ;
   format Country $9. ;
   format Amount_Billed_in_US_Dollars best12. ;
   format Billed_By best12. ;
   format Billed_On $9. ;
   format Paid_On date9. ;
   do;
      EFIOUT + 1;
      put Invoice_ID @;
      put Billed_To @;
      put Amount_Billed_in_Local_Currency @;
      put Country $ @;
      put Amount_Billed_in_US_Dollars @;
      put Billed_By @;
      put Billed_On $ @;
      put Paid_On ;
   end;
if _ERROR_ then call symputx('_EFIERR_','1'); /* set ERROR detection macro variable */
if EFIEOD then call symputx('_EFIREC_','EFIOUT');
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=29Mar2016:16:34:34

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

PROC PRINT;
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

OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
Chapter 7
FORMAT Procedure

Overview: FORMAT Procedure ........................................ 96
   What Does the FORMAT Procedure Do? ......................... 96
   Format Catalog Encodings in SAS Viya .......................... 96
   What Are Formats and Informat\(s\)? ............................. 96
   How Are Formats and Informat\(s\) Associated with a Variable? 97

Concepts: FORMAT Procedure ....................................... 97
   Associating Informats and Formats with Variables .............. 97
   Storing Informats and Formats ................................... 98
   Printing Informats and Formats .................................. 100
   Using Formats in a CAS Session ................................. 101
   A Binary Search Determines the User-Defined Format or Informat for a Value 101

Syntax: FORMAT Procedure .......................................... 102
   PROC FORMAT Statement .......................................... 103
   EXCLUDE Statement .............................................. 107
   INVALUE Statement ............................................... 108
   PICTURE Statement ............................................... 113
   SELECT Statement ................................................ 132
   VALUE Statement .................................................. 133

Specifying Values or Ranges ........................................ 139

Results: FORMAT Procedure .......................................... 141
   Output Control Data Set ......................................... 141
   Procedure Output ................................................. 144

Examples: FORMAT Procedure ....................................... 147
   Example 1: Create a Format Library in a CAS Session ........ 147
   Example 2: Create the Example Data Set ........................ 151
   Example 3: Creating a Picture Format ............................ 152
   Example 4: Creating a Picture Format for Large Dollar Amounts 154
   Example 5: Filling a Picture Format .............................. 156
   Example 6: Creating a Format for Character Values ............ 158
   Example 7: Creating a Format for Missing and Nonmissing Variable Values 160
   Example 8: Creating a Format from a Data Set .................. 163
   Example 9: Retrieving a Permanent Format ...................... 166
   Example 10: Creating a Format in a non-English Language .... 169
   Example 11: Using a Format to Create a Drill-down Table ..... 171
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 “Converting Format Catalogs to UTF-8” in SAS Viya National Language Support (NLS): Reference Guide.

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\textit{w.d} informat and the DOLLAR\textit{w.d} format are provided by SAS.

\textit{Figure 7.1}  Associating an Informat and a Format with a Variable

raw data value \textdollar 1,544.32

\begin{itemize}
  \item read with COMMA9.2 informat
  \item converted value 1544.32
  \item printed using DOLLAR9.2 format
  \item printed value \textdollar 1,544.32
\end{itemize}

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 97.

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 7.1  Associating Informats and Formats with Variables

<table>
<thead>
<tr>
<th>Step</th>
<th>Informats</th>
<th>Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>In a DATA step</td>
<td>Use the ATTRIB or INFORMAT statement to permanently associate an informat</td>
<td>Use the ATTRIB or FORMAT statement to permanently associate a format with</td>
</tr>
<tr>
<td></td>
<td>with a variable. Use the INPUT function or INPUT statement to associate</td>
<td>a variable. Use the PUT function or PUT statement to associate the format</td>
</tr>
<tr>
<td></td>
<td>the informat with the variable only for the duration of the DATA step.</td>
<td>with the variable only for the duration of the DATA step.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In a PROC step</td>
<td>The ATTRIB and INFORMAT statements are valid in SAS procedures. However,</td>
<td>Use the ATTRIB statement or the FORMAT statement to associate formats</td>
</tr>
<tr>
<td></td>
<td>in SAS software, typically you do not assign informats in PROC steps</td>
<td>with variables. If you use either statement in a procedure that produces</td>
</tr>
<tr>
<td></td>
<td>because the data has already been read into SAS variables.</td>
<td>an output data set, then the format is permanently associated with the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable in the output data set. If you use either statement in a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>procedure that does not produce an output data set or modify an existing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data set, the statement associates the format with the variable only for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the duration of the PROC step.</td>
</tr>
</tbody>
</table>

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.

Footnotes:
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 103.

**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

```sas
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:

```sas
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 104.

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:

```plaintext
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:

```plaintext
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.

Informatss cannot be written to a CAS session. Informatss included in your SAS code are ignored.

Formats that use functions-as-labels or formats-as-labels cannot be written to CAS.

SAS Viya supports only the UTF-8 encoding. For information about the encodings of format catalogs, see “Converting Format Catalogs to UTF-8” in SAS Viya National Language Support (NLS): Reference Guide.

**Tip:** You can also use appropriate global statements with this procedure.

**See:** “Using Formats in a CAS Session” on page 101

For information about using VARCHAR, see “Create a VARCHAR Variable Using the LENGTH Statement” in SAS Cloud Analytic Services: Accessing and Manipulating Data and “Index CHAR and VARCHAR Character Strings” in SAS Cloud Analytic Services: Accessing and Manipulating Data.

**PROC FORMAT**

```plaintext
PROC FORMAT <option(s)>;
  EXCLUDE entry(s);
  INV ALUE <$>name <(informat-option(s))> <value-range-set(s)>;
  PICTURE name <(format-option(s))>
    <value-range-set-1 <(picture-1-option(s))>
    <value-range-set-2 <(picture-2-option(s))>> ...>;
  SELECT entry(s);
  VALUE <$>name <(format-option(s))> <value-range-set(s)>;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC FORMAT</td>
<td>Define formats and informats for variables</td>
<td>Ex. 3, Ex. 8, Ex. 9</td>
</tr>
<tr>
<td>EXCLUDE</td>
<td>Exclude catalog entries from processing by the FMTLIB and CNTLOUT= options</td>
<td></td>
</tr>
<tr>
<td>INV ALUE</td>
<td>Create an informat for reading and converting raw data values</td>
<td></td>
</tr>
<tr>
<td>PICTURE</td>
<td>Create a template for printing numbers</td>
<td>Ex. 3, Ex. 5, Ex. 10</td>
</tr>
<tr>
<td>SELECT</td>
<td>Select catalog entries for processing by the FMTLIB and CNTLOUT= options</td>
<td></td>
</tr>
</tbody>
</table>
PROC FORMAT Statement

Creates user-specified formats and informats for variables.

**Tip:** You can use data set options with the CNTLIN= and CNTLOUT= data set options.

**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 152
- “Example 8: Creating a Format from a Data Set” on page 163
- “Example 9: Retrieving a Permanent Format” on page 166

**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.

- **MAXLABELN=** *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.
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 INV ALUE 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.
- 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

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 101
- “Example 1: Create a Format Library in a CAS Session” on page 147
- “Using Formats in a CAS Session” on page 101

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 163

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 141

SAS Viya supports only UTF-8 encoding. For information about the encoding of your format catalogs in SAS Viya, see “Converting Format Catalogs to UTF-8” in SAS Viya National Language Support (NLS): Reference Guide.

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 98

Example  “Example 3: Creating a Picture Format” on page 152

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 (NLS): 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;
```

MAXLABELN=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: Informs 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.
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.

<table>
<thead>
<tr>
<th>Defaults</th>
<th>For character informats, the length of the longest label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For numeric informats, 12 if you have numeric data to the left of the equal sign</td>
</tr>
<tr>
<td></td>
<td>For quoted strings, the length of the longest string</td>
</tr>
</tbody>
</table>

| 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:

```plaintext
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.

**REEXP**
**REGEPE**

specifies that the preceding range is to be treated as a Perl regular expression. If you specify REGEPE, 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 REEXP option are the same as they are for the PRXPARE function in the DATA step. The rules for the REGEPE 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 REEXP and E for REGEPE.

**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 139.

**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 informated value. Typically, *number* becomes the value of a numeric variable when you use the informat to convert raw data. Use *number* for *formattted-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.

```plaintext
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':

```plaintext
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:

```plaintext
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:

```sas
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 134.
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 (NLS): Reference Guide for documentation about formats that are supplied by SAS.

**Examples:**
"Example 3: Creating a Picture Format" on page 152
"Example 5: Filling a Picture Format" on page 156
"Example 10: Creating a Format in a non-English Language" on page 169

**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 121 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 156

**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:

<table>
<thead>
<tr>
<th>Language</th>
<th>English</th>
<th>Macedonian</th>
<th>Spanish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afrikaans</td>
<td>English</td>
<td>Macedonian</td>
<td>Spanish</td>
</tr>
<tr>
<td>Catalan</td>
<td>Finnish</td>
<td>Norwegian</td>
<td>Swedish</td>
</tr>
<tr>
<td>Croatian</td>
<td>French</td>
<td>Polish</td>
<td>Swiss_French</td>
</tr>
<tr>
<td>Czech</td>
<td>German</td>
<td>Portuguese</td>
<td>Swiss_German</td>
</tr>
<tr>
<td>Danish</td>
<td>Hungarian</td>
<td>Russian</td>
<td></td>
</tr>
<tr>
<td>Dutch</td>
<td>Italian</td>
<td>Slovenian</td>
<td></td>
</tr>
</tbody>
</table>
Defaults

For single-byte character sets, the language that is specified by the 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 (NLS): Reference Guide.

See


**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.

```plaintext
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=*

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:

```plaintext
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.
<table>
<thead>
<tr>
<th>Restriction</th>
<th>The MULT= option is not valid when you use a function to format a value.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
<th>“Example 3: Creating a Picture Format” on page 152</th>
</tr>
</thead>
</table>

| Example 4: Creating a Picture Format for Large Dollar Amounts” on page 154 |

<table>
<thead>
<tr>
<th>NOEDIT</th>
<th>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 &gt;1000 miles:</th>
</tr>
</thead>
</table>
|        | picture miles 1-1000='0000'
|        | 1000<-high='>1000 miles'(noedit); |

<table>
<thead>
<tr>
<th>Restriction</th>
<th>The NOEDIT= option is not valid when you use a function to format a value.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>NOTSORTED</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use NOTSORTED if one of the following is true:</td>
<td></td>
</tr>
<tr>
<td>• You know the likelihood of certain ranges occurring, and you want your format to search those ranges first to save processing time.</td>
<td></td>
</tr>
<tr>
<td>• You want to preserve the order that you define ranges when you print a description of the format using the FMTLIB option.</td>
<td></td>
</tr>
<tr>
<td>• 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.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>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.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>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:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Use the CNTLOUT= option in the PROC FORMAT statement to create an output control data set.</td>
</tr>
<tr>
<td>• Use the CPORT procedure to create a transport file for the control data set.</td>
</tr>
<tr>
<td>• Use the CIMPORT procedure in the target operating environment to import the transport file.</td>
</tr>
<tr>
<td>• In the target operating environment, use PROC FORMAT with the CNTLIN= option to build the formats and informats from the imported control data set.</td>
</tr>
<tr>
<td>---------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
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 152
- “Example 5: Filling a Picture Format” on page 156

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'
```

See “Specifying Values or Ranges” on page 139.
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 115 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:

```sas
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;
```
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 <  0 = '99' (prefix='.' mult=100)
  0.99 < - < 1 = '99' (prefix='.' 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 <  0 = '99' (prefix='.' mult=100)
  0.99 < - < 1 = '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 7.3 on page 129.
Table 7.2 Building a Picture Format

<table>
<thead>
<tr>
<th>Step</th>
<th>Rules for Processing the PICTURE Statement</th>
<th>In This Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Determine into which range the value falls and use that picture.</td>
<td>In the second range, the exclusion operator &lt; 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.</td>
</tr>
<tr>
<td>2</td>
<td>Take the absolute value of the numeric value.</td>
<td>Because the absolute value is used, you need a separate range and picture for the negative numbers in order to prefix the minus sign.</td>
</tr>
<tr>
<td>3</td>
<td>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.)</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>4a</td>
<td>When the ROUND option is not performed, the numbers after the decimal are truncated.</td>
<td>205.1 is truncated to 205. 55.6 is truncated to 55. 99.6 is truncated to 99.</td>
</tr>
<tr>
<td>5</td>
<td>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 Zw. format. Zw. is explained in the section on SAS formats in SAS Viya Formats and Informats: Reference.)</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Step 5a

**Rules for Processing the PICTURE Statement**

- 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.

### Step 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 rightmost \( n \) 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.
### Rules for Processing the PICTURE Statement

In This Example

- **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.

#### 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 7.3 Steps to Format Various Values

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>-2.051</th>
<th>-.05</th>
<th>.556</th>
<th>.996</th>
<th>6.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Range</td>
<td>low – –1</td>
<td>–0.99 &lt; – &lt; 0</td>
<td>0 &lt; – &lt; .99</td>
<td>0.99 – &lt; 1</td>
<td>1 – high</td>
</tr>
<tr>
<td></td>
<td>Picture</td>
<td>000.00</td>
<td>99</td>
<td>99</td>
<td>0.99</td>
<td>09.99</td>
</tr>
<tr>
<td>2</td>
<td>Absolute value</td>
<td>2.051</td>
<td>.05</td>
<td>.556</td>
<td>.996</td>
<td>6.6</td>
</tr>
<tr>
<td>3</td>
<td>MULT</td>
<td>2.051×10²=205.1</td>
<td>.05×100=5</td>
<td>.556×100=55.6</td>
<td>.996×100=99.6</td>
<td>6.6×10²=660</td>
</tr>
<tr>
<td>4</td>
<td>Round</td>
<td>205</td>
<td>5</td>
<td>56</td>
<td>100</td>
<td>660</td>
</tr>
<tr>
<td>4a</td>
<td>No Rounding</td>
<td>205</td>
<td>5</td>
<td>55</td>
<td>99</td>
<td>660</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>-2.051</th>
<th>-05</th>
<th>.556</th>
<th>.996</th>
<th>6.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Character string, rounding</td>
<td>00205</td>
<td>05</td>
<td>56</td>
<td>100</td>
<td>0660</td>
</tr>
<tr>
<td>5a</td>
<td>Character string, no rounding</td>
<td>00205</td>
<td>05</td>
<td>55</td>
<td>099</td>
<td>0660</td>
</tr>
<tr>
<td>6</td>
<td>Template, rounding</td>
<td>2.05</td>
<td>05</td>
<td>56</td>
<td>1.00</td>
<td>6.60</td>
</tr>
<tr>
<td>6a</td>
<td>Template, no rounding</td>
<td>2.05</td>
<td>05</td>
<td>55</td>
<td>99</td>
<td>6.60</td>
</tr>
<tr>
<td>7</td>
<td>Prefix, rounding</td>
<td>prefix='-'</td>
<td>prefix='--'</td>
<td>prefix='.'</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>7a</td>
<td>Prefix, no rounding</td>
<td>prefix='-'</td>
<td>prefix='--'</td>
<td>prefix='.'</td>
<td>prefix='.'</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>Formatted result, rounding</td>
<td>−2.05</td>
<td>−05</td>
<td>.56</td>
<td>1.00</td>
<td>6.60</td>
</tr>
<tr>
<td></td>
<td>Formatted results, no rounding</td>
<td>−2.05</td>
<td>−05</td>
<td>.55</td>
<td>.99</td>
<td>6.60</td>
</tr>
</tbody>
</table>

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.

```plaintext
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

<table>
<thead>
<tr>
<th>Obs</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-45.00</td>
</tr>
<tr>
<td>2</td>
<td>-2.05</td>
</tr>
<tr>
<td>3</td>
<td>-1.00</td>
</tr>
<tr>
<td>4</td>
<td>-.05</td>
</tr>
<tr>
<td>5</td>
<td>-.02</td>
</tr>
<tr>
<td>6</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>.09</td>
</tr>
<tr>
<td>8</td>
<td>.54</td>
</tr>
<tr>
<td>9</td>
<td>.56</td>
</tr>
<tr>
<td>10</td>
<td>1.00</td>
</tr>
<tr>
<td>11</td>
<td>6.60</td>
</tr>
<tr>
<td>12</td>
<td>14.63</td>
</tr>
</tbody>
</table>

### Formatting the Variable Amount with the NOZEROS. Format, No Rounding

<table>
<thead>
<tr>
<th>Obs</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-45.00</td>
</tr>
<tr>
<td>2</td>
<td>-2.05</td>
</tr>
<tr>
<td>3</td>
<td>-.99</td>
</tr>
<tr>
<td>4</td>
<td>-.05</td>
</tr>
<tr>
<td>5</td>
<td>-.01</td>
</tr>
<tr>
<td>6</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>.09</td>
</tr>
<tr>
<td>8</td>
<td>.54</td>
</tr>
<tr>
<td>9</td>
<td>.55</td>
</tr>
<tr>
<td>10</td>
<td>.99</td>
</tr>
<tr>
<td>11</td>
<td>6.60</td>
</tr>
<tr>
<td>12</td>
<td>14.63</td>
</tr>
</tbody>
</table>
**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.

```sql
select a:;
```

In addition, the following SELECT statement selects all formats or informats that occur alphabetically between `apple` and `pear`, inclusive:

```sql
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:

```sql
ERROR: User does not have appropriate authorization level for file libref.FORMATS.CATALOG.
```

**VALUE Statement**

**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'
```
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:

\[ \text{value-or-range-1} <, \text{value-or-range-2}, \ldots>=\text{existing-format} | \text{[existing-format]} \]

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 139.

**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:

```plaintext
value $ score 
'M'="Male"
'F'="Female";
```

If a formatted value contains a single quotation mark, then enclose the value in double quotation marks:

```plaintext
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 134.

Examples

Example 1: Create a Format to Print Postal Codes for Selected States
The `$STATE.` character format prints the postal code for selected states:

```plaintext
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:

```plaintext
value answer 1='yes'
    2='no';
```

Example 3: Specifying No Ranges
This VALUE statement creates a `format-name` format that has no ranges:

```plaintext
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:

```plaintext
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. */
```
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 158.

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 \textit{AJ} is part of the first range:

```
'AA'-'AJ'=1 'AJ'-'AZ'=2
```

In this example, to include the value \textit{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 \texttt{NORMAL}:

```
value temp 98.6='NORMAL';
```

If the value were 96.9, then the printed result would be \texttt{96.9}.
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 programatically 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 7.1  Output Control Data Set for PROC FORMAT Examples
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 158.
The fields are described below in the order in which they appear in the output, from left to right:

**FORMAT 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 \textbf{M} to represent each option, followed by the value. For example, in the LABEL section, \textbf{P-} indicates that the prefix value is a hyphen followed by a period.

FMTLIB prints only 40 characters in the LABEL column.

---

\textbf{Examples: FORMAT Procedure}

---

\textbf{Example 1: Create a Format Library in a CAS Session}

\textbf{Features:}

\begin{itemize}
  \item PROC FORMAT statement option
  \item CASFMTLIB
  \item CAS statement
\end{itemize}

\textbf{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.

\textbf{Program}

```sas
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;
Example 1: Create a Format Library in a CAS Session

Log 7.1  Create a Format Library in a CAS Session

```
OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
libname proclib cas;
NOTE: Libref was successfully assigned as follows:
   Engine:       CAS
   Physical Name: 5f54c1ea-4c0c-3b42-876e-ace7d5651f2b
proc format casfmlib='myformats';
 NOTE: Both CAS-based formats and catalog-based formats will be written.
! value hospx 1='New_York' 2='Massachusetts_General' 3='Los_Angeles'
4='Mary_Fletcher';
NOTE: Format HOSPX is already on the library WORK.FORMATS.
NOTE: Format HOSPX has been output.
run;

NOTE: PROCEDURE FORMAT used (Total process time):
   real time           0.02 seconds
   cpu time            0.00 seconds

data clinicalTrial;
input hospital treatment $ @@;
severity=rannor(1323)*5 + 10;
format hospital hospx.;
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
;
   data proclib.clinicalTrial;
   set work.clinicalTrial;
   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

proc regselect data=proclib.clinicalTrial;
   class treatment hospital;
   model severity=treatment hospital;
   run;

NOTE: The Cloud Analytic Services server processed the request in 0.01923 seconds.
NOTE: PROCEDURE RESELECTION used (Total process time):
   real time           0.41 seconds
   cpu time            0.10 seconds

OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```
Creating a Format Library in a CAS Session

The output is divided into sections only for documentation appearances.
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 141 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# 21161# 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
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';
Example 3: Creating a Picture Format

```sas
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;
```
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 display 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 120.

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;
```

Output

Output 7.3 PROCLIB.STAFF with a Format for the Variable Salary

<table>
<thead>
<tr>
<th>Name</th>
<th>IDNumber</th>
<th>Salary in U.S. Dollars</th>
<th>Site</th>
<th>HireDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capalletti, Jimmy</td>
<td>2355</td>
<td>$34,072</td>
<td>BR1</td>
<td>30JAN13</td>
</tr>
<tr>
<td>Chen, Len</td>
<td>5889</td>
<td>$33,771</td>
<td>BR1</td>
<td>18JUN08</td>
</tr>
<tr>
<td>Davis, Brad</td>
<td>3878</td>
<td>$31,500</td>
<td>BR2</td>
<td>20MAR04</td>
</tr>
<tr>
<td>Leung, Brenda</td>
<td>4409</td>
<td>$55,256</td>
<td>BR2</td>
<td>18SEP94</td>
</tr>
<tr>
<td>Martinez, Maria</td>
<td>3965</td>
<td>$78,980</td>
<td>US1</td>
<td>10JAN93</td>
</tr>
<tr>
<td>Orfali, Philip</td>
<td>0740</td>
<td>$80,648</td>
<td>US2</td>
<td>18FEB03</td>
</tr>
<tr>
<td>Patel, Mary</td>
<td>2398</td>
<td>$56,643</td>
<td>BR3</td>
<td>02FEB90</td>
</tr>
<tr>
<td>Smith, Robert</td>
<td>5162</td>
<td>$84,561</td>
<td>BR5</td>
<td>15APR06</td>
</tr>
<tr>
<td>Sorrell, Joseph</td>
<td>4421</td>
<td>$82,403</td>
<td>US1</td>
<td>19JUN11</td>
</tr>
<tr>
<td>Zook, Carla</td>
<td>7385</td>
<td>$37,010</td>
<td>BR3</td>
<td>18DEC10</td>
</tr>
</tbody>
</table>
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.

Example 4: Creating a Picture Format for Large Dollar Amounts

```
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 7.2 Formatted Millions, Billions, and Trillions Dollar Amounts

<table>
<thead>
<tr>
<th>x</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,048,576</td>
<td>$1 M</td>
</tr>
<tr>
<td>16,777,216</td>
<td>$16 M</td>
</tr>
<tr>
<td>268,435,456</td>
<td>$268 M</td>
</tr>
<tr>
<td>4,294,967,296</td>
<td>$4 B</td>
</tr>
<tr>
<td>68,719,476,736</td>
<td>$68 B</td>
</tr>
<tr>
<td>1,099,511,627,776</td>
<td>$1 T</td>
</tr>
<tr>
<td>17,592,186,044,416</td>
<td>$17 T</td>
</tr>
<tr>
<td>281,474,976,710,656</td>
<td>$281 T</td>
</tr>
</tbody>
</table>

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 7.3 More Precisely Formatted Large Dollar Amounts

<table>
<thead>
<tr>
<th>x</th>
<th>$x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,048,576</td>
<td>$1.04 M</td>
</tr>
<tr>
<td>16,777,216</td>
<td>$16.77 M</td>
</tr>
<tr>
<td>268,435,456</td>
<td>$268.43 M</td>
</tr>
<tr>
<td>4,294,967,296</td>
<td>$4.29 B</td>
</tr>
<tr>
<td>68,719,476,736</td>
<td>$68.71 B</td>
</tr>
<tr>
<td>1,099,511,627,776</td>
<td>$1.09 T</td>
</tr>
<tr>
<td>17,592,186,044,416</td>
<td>$17.59 T</td>
</tr>
<tr>
<td>281,474,976,710,656</td>
<td>$281.47 T</td>
</tr>
</tbody>
</table>

Example 5: Filling a Picture Format

Features: PICTURE statement options
  FILL=...
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

```plaintext
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.

```plaintext
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 leftmost comma in the picture is replaced by the fill character.

```plaintext
proc format;
```
picture salary low-high='00,000000.00' {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.;
run;

Specify the title.

title 'Printing Salaries for a Check';
run;

Output

Output 7.4  Printing Salaries for a Check

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.
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**

```sas
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 \textit{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 7.6  Dropout Rates and Math Scores for Each State in a Region**

#### The SAS System

**Region=MW**

<table>
<thead>
<tr>
<th>Obs</th>
<th>State</th>
<th>DropoutRate</th>
<th>MathScore</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Illinois</td>
<td>21.5</td>
<td>260.0</td>
</tr>
<tr>
<td>2</td>
<td>Indiana</td>
<td>13.8</td>
<td>267.0</td>
</tr>
<tr>
<td>3</td>
<td>Iowa</td>
<td>13.6</td>
<td>278.0</td>
</tr>
<tr>
<td>4</td>
<td>Kansas</td>
<td>17.9</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**Region=NE**

<table>
<thead>
<tr>
<th>Obs</th>
<th>State</th>
<th>DropoutRate</th>
<th>MathScore</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Connecticut</td>
<td>16.8</td>
<td>270.0</td>
</tr>
<tr>
<td>6</td>
<td>Delaware</td>
<td>28.5</td>
<td>261.0</td>
</tr>
<tr>
<td>7</td>
<td>Maine</td>
<td>22.5</td>
<td>n/a</td>
</tr>
<tr>
<td>8</td>
<td>Maryland</td>
<td>26.0</td>
<td>260.0</td>
</tr>
</tbody>
</table>

**Region=SE**

<table>
<thead>
<tr>
<th>Obs</th>
<th>State</th>
<th>DropoutRate</th>
<th>MathScore</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Alabama</td>
<td>22.3</td>
<td>252.0</td>
</tr>
<tr>
<td>10</td>
<td>Arkansas</td>
<td>11.5</td>
<td>256.0</td>
</tr>
<tr>
<td>11</td>
<td>Florida</td>
<td>36.5</td>
<td>255.0</td>
</tr>
<tr>
<td>12</td>
<td>Georgia</td>
<td>27.9</td>
<td>258.0</td>
</tr>
<tr>
<td>13</td>
<td>Kentucky</td>
<td>32.7</td>
<td>256.0</td>
</tr>
<tr>
<td>14</td>
<td>Louisiana</td>
<td>43.1</td>
<td>246.0</td>
</tr>
</tbody>
</table>

**Region=W**

<table>
<thead>
<tr>
<th>Obs</th>
<th>State</th>
<th>DropoutRate</th>
<th>MathScore</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Alaska</td>
<td>35.8</td>
<td>n/a</td>
</tr>
<tr>
<td>16</td>
<td>Arizona</td>
<td>31.2</td>
<td>259.0</td>
</tr>
<tr>
<td>17</td>
<td>California</td>
<td>32.7</td>
<td>256.0</td>
</tr>
<tr>
<td>18</td>
<td>Colorado</td>
<td>24.7</td>
<td>267.0</td>
</tr>
<tr>
<td>19</td>
<td>Hawaii</td>
<td>18.3</td>
<td>251.0</td>
</tr>
<tr>
<td>20</td>
<td>Idaho</td>
<td>21.8</td>
<td>272.0</td>
</tr>
</tbody>
</table>
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**
```sas
libname proclib cas;
data scale;
imput 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 0 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='0';
  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 7.7 The CTRL Data Set

<table>
<thead>
<tr>
<th>label</th>
<th>start</th>
<th>end</th>
<th>fmtname</th>
<th>type</th>
<th>hlo</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0</td>
<td>3</td>
<td>PercentageFormat</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>3%</td>
<td>4</td>
<td>6</td>
<td>PercentageFormat</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>6%</td>
<td>7</td>
<td>8</td>
<td>PercentageFormat</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>8%</td>
<td>9</td>
<td>10</td>
<td>PercentageFormat</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>10%</td>
<td>11</td>
<td>16</td>
<td>PercentageFormat</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td><em><strong>ERROR</strong></em></td>
<td>11</td>
<td>16</td>
<td>PercentageFormat</td>
<td>n</td>
<td>O</td>
</tr>
</tbody>
</table>

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
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 125.

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

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;
Print the SAMPLE data set. The FORMAT statement associates the NOZEROS. format with the Amount variable.

```plaintext
proc print data=sample;
    format amount nozeros.;
run;
```

Specify the titles.

```plaintext
title1 'Retrieving the NOZEROS. Format from PROCLIB.FORMATS';
title2 'The SAMPLE Data Set';
run;
```

Output

**Output 7.8**  Retrieving the NOZEROS. Format from PROCLIB.FORMATS

### The SAMPLE Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2.05</td>
</tr>
<tr>
<td>2</td>
<td>0.05</td>
</tr>
<tr>
<td>3</td>
<td>0.01</td>
</tr>
<tr>
<td>4</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>0.09</td>
</tr>
<tr>
<td>6</td>
<td>0.54</td>
</tr>
<tr>
<td>7</td>
<td>0.55</td>
</tr>
<tr>
<td>8</td>
<td>6.60</td>
</tr>
<tr>
<td>9</td>
<td>14.63</td>
</tr>
<tr>
<td>10</td>
<td>0.99</td>
</tr>
<tr>
<td>11</td>
<td>0.99</td>
</tr>
<tr>
<td>12</td>
<td>-45.00</td>
</tr>
</tbody>
</table>
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 %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 113.

libname proclib cas;
proc format;
  picture mdy(default=8) other='%d%m%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 LANGSTSFR. 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

Example 11: Using a Format to Create a Drill-down Table

Features:
- VALUE statement

Other features:
- PROC PRINT FORMAT statement

---

Example 11: Using a Format to Create a Drill-down Table

```sas
57 libname proclib cas;
58  
59     proc format ;
60  
61       !  picture mdy(default=8) other='%d%m%Y' (datatype=date);
NOTE: Format MDY has been output.
62       !  picture langtsda (default=50) other='%A, %d %B, %Y' (datatype=date);
NOTE: Format LANGTSDA has been output.
63       !  picture langtsdt (default=50) other='%A, %d,%B, %Y %H %M %S'
       (datatype=datetime);
NOTE: Format LANGTSDT has been output.
64       !  picture langtsfr (default=50) other='%A, %d %B, %Y %H %M %S'
       (datatype=datetime language=french);
NOTE: Format LANGTSFR has been output.
65       !  picture alltest (default=100)
       other='%a %A %b %B %d %H %I %j %m %M %p %S %w %U %y %%'
       (datatype=datetime);
NOTE: Format ALLTEST has been output.
66     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 %
```
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 (\&lt;a\&gt;) 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 SCOMPND. 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 7.9** Using a Format to Create Drill-down Text in an HTML Table

![Year 2000 U.S. Census Population](image)

(Click the underlined text to drill down.)
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**

```plaintext
PROC HTTP URL="URL-to-target" <option(s)>;
   HEADERS "HeaderName"="HeaderValue" "HeaderName-n"="HeaderValue-n";
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC HTTP</td>
<td>Issue HTTP requests</td>
<td>Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7, Ex. 8, Ex. 9, Ex. 10, Ex. 11</td>
</tr>
<tr>
<td>HEADERS</td>
<td>Specify request headers for the HTTP request</td>
<td>Ex. 9</td>
</tr>
</tbody>
</table>

### PROC HTTP Statement

Invokes a web service that issues requests.

**Examples:**

- "Example 1: A Simple GET Request" on page 184
- "Example 2: A Simple PUT Request" on page 184
- "Example 3: A Simple Post Request Using TLS" on page 185
- "Example 4: Specifying Input Data as a String" on page 185
- "Example 5: A Proxy Set in a Macro Variable" on page 186
- "Example 6: A Proxy Specified in the HTTP Request" on page 187
- "Example 7: A POST That Captures the Response Headers" on page 187
- "Example 8: A GET That Specifies HEADEROUT_OVERWRITE" on page 188
- "Example 9: A GET That Uses the HEADERS Statement" on page 189
- "Example 10: A Nonstandard Method" on page 190
- "Example 11: A PUT That Specifies EXPECT_100_CONTINU" on page 191

### Syntax

**PROC HTTP**

```plaintext
PROC HTTP URL="URL-to-target" 
   <METHOD="http-method"> 
   <authentication-type-options>
   <caching-options>
   <header-options>
```
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 191

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
187

“Example 11: A PUT That Specifies EXPECT_100_CONTINUE” on
page 191

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 188
**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 185

**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 184

“Example 9: A GET That Uses the HEADERS Statement” on page 189

“Example 10: A Nonstandard Method” on page 190

**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 184

**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 187

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 189

Syntax

HEADERS "HeaderValue"="HeaderValue" <"HeaderName-n"="HeaderValue-n">;

Required Argument

"HeaderValue"="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.
<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Default Content-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>application/x-www-form-urlencoded</td>
</tr>
<tr>
<td>PUT</td>
<td>application/octet-stream</td>
</tr>
</tbody>
</table>

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*. Also see “Manage Certificates for TLS and HTTPS” in *Encryption in SAS Viya*. “Example 3: A Simple Post Request Using TLS” on page 185 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 186.

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;
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
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

```sas
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

```sas
filename resp TEMP;

proc http
  url="http://httpbin.org/post"
  in='custname=Sas+User&custtel=919-555-5555&custemail=sas.user@sas.com&size=medium&topping=cheese&delivery=12%3A00&comments=Dont+Drop+It'
run;
```
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**

```sas
%let PROCHTTP_PROXY="http://myproxy:889";

filename out "u:\prochttp\Testware\ProxyTest_out.txt";
```
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 186.

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 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 186 but captures the response headers in a file called headerOut.txt.

Program

%let PROCHTTP_PROXY="http://myproxy:889";
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

```plaintext
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:
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

```plaintext
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

```plaintext
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**

```plaintext
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"  
}
Overview: IMPORT Procedure

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” in SAS Viya National Language Support (NLS): Reference Guide.

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.

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

If you use this method and 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.

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 outputs the correct value based on the specified locale. If you input 14,000.01 with NLNUM and LOCALE=French_France, store it in a data set, and then output 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
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" in SAS Viya National Language Support (NLS): Reference Guide.

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;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC IMPORT</td>
<td>Import an external data file to a SAS data set</td>
<td>Ex. 1, Ex. 2, Ex. 3, Ex. 4</td>
</tr>
<tr>
<td>DATAROW</td>
<td>Start reading data from a specific row in the delimited text file</td>
<td>Ex. 3</td>
</tr>
<tr>
<td>DBENCODING</td>
<td>Indicate the encoding character set to use for the JMP file</td>
<td></td>
</tr>
<tr>
<td>DELIMITER</td>
<td>Specify the delimiter that separates columns of data in the input file</td>
<td>Ex. 1, Ex. 3, Ex. 4</td>
</tr>
</tbody>
</table>
### 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 202
- “Example 2: Importing a Specific Delimited File Using a Fileref” on page 205
- “Example 3: Importing a Tab-Delimited File” on page 209
- “Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 212

**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 202
- “Example 2: Importing a Specific Delimited File Using a Fileref” on page 205
- “Example 3: Importing a Tab-Delimited File” on page 209
“Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 212

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 6.1 on page 76 for more information about identifiers for this option.

Examples

“Example 1: Importing a Delimited File” on page 202

“Example 2: Importing a Specific Delimited File Using a Fileref” on page 205

“Example 3: Importing a Tab-Delimited File” on page 209

“Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 212

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 output 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 202

“Example 2: Importing a Specific Delimited File Using a Fileref” on page 205

“Example 3: Importing a Tab-Delimited File” on page 209

“Example 4: Importing a Comma-Delimited File with a CSV Extension” on page 212

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 201

Example:

“Example 3: Importing a Tab-Delimited File” on page 209

**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 202

**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 202
"Example 2: Importing a Specific Delimited File Using a Fileref" on page 205
"Example 4: Importing a Comma-Delimited File with a CSV Extension" on page 212

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 change the default row value in the SAS Registry. From the SAS command line, enter regedit. When the Registry Editor opens, select Products ⇒ BASE ⇒ EFI ⇒ GuessingRows.

MAX
can be specified instead of 2147483647. Specifying the maximum value could adversely affect performance.

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&4000&10000
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 9.1 External File Imported to Create a SAS Data Set

```sas
options nodate ps=60 ls=80;
proc import datafile="/userid/pathname/delimiter.txt"
  dbms=dlm
  out=mydata
  replace;
  delimiter='&';
  getnames=yes;
run;

/***************************************************************
***
* PRODUCT: SAS
* VERSION: V.03.00
* CREATOR: External File Interface
* DATE: 25MAR16
* DESC: Generated SAS Datastep Code
* TEMPLATE SOURCE: (None Specified.)
***************************************************************
***
data WORK.MYDATA;
%let _EFIERR_ = 0; /* set the ERROR detection macro variable */
infile '/u/userid/pathname/delimiter.txt' delimiter = "&" MISSOVER DSD lrecl=32767 firstobs=2;
informat Region $8.;
informat State $2.;
informat Month MONYY7.;
informat Expenses best32.;
informat Revenue best32.;
informat Region $8.;
informat State $2.;
informat Month MONYY7.;
informat Expenses best32.;
informat Revenue best32.;
input
  Region $;
  State $;
  Month
  Expenses
  Revenue
;           
if _ERROR_ then call symputx('_EFIERR_','1'); /* set ERROR detection macro variable */
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=25Mar2016:11:22:03,
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.

8 rows created in WORK.MYDATA from /u/userid/pathname/delimiter.txt.

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.

Output 9.1  Data Set Work.MyData

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>Region</th>
<th>State</th>
<th>Month</th>
<th>Expenses</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Southern</td>
<td>GA</td>
<td>JAN2001</td>
<td>2000</td>
<td>8000</td>
</tr>
<tr>
<td>2</td>
<td>Southern</td>
<td>GA</td>
<td>FEB2001</td>
<td>1200</td>
<td>6000</td>
</tr>
<tr>
<td>3</td>
<td>Southern</td>
<td>FL</td>
<td>FEB2001</td>
<td>8500</td>
<td>11000</td>
</tr>
<tr>
<td>4</td>
<td>Northern</td>
<td>NY</td>
<td>FEB2001</td>
<td>3000</td>
<td>4000</td>
</tr>
<tr>
<td>5</td>
<td>Northern</td>
<td>NY</td>
<td>MAR2001</td>
<td>6000</td>
<td>5000</td>
</tr>
<tr>
<td>6</td>
<td>Southern</td>
<td>FL</td>
<td>MAR2001</td>
<td>9800</td>
<td>13500</td>
</tr>
<tr>
<td>7</td>
<td>Northern</td>
<td>MA</td>
<td>MAR2001</td>
<td>1500</td>
<td>1000</td>
</tr>
</tbody>
</table>

Example 2: Importing a Specific Delimited File Using a Fileref

Features: PROC IMPORT statement options
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' lrec=100;

proc import datafile=stda
   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' lrec=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=stda
   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 9.2  Importing a Specific Delimited File Using a Fileref

```sas
OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
filename stdata '/userid/pathname/state_data.txt' lrecl=100;
proc import datafile=stdata dbms=dlm out=states replace;
delimiter=' ';
getnames=yes;
run;

/*****************************************************************
*   PRODUCT:   SAS
*   VERSION:   V.03.00
*   CREATOR:   External File Interface
*   DATE:      25MAR16
*   DESC:      Generated SAS Datastep Code
*   TEMPLATE SOURCE:  (None Specified.)
*****************************************************************/
data WORK.STATES    ;
&let _EFIERR_ = 0; /* set the ERROR detection macro variable */
infile STDATA delimiter = ' ' MISSOVER DSD  firstobs=2 ;
   informat Region $7. ;
   informat State $16. ;
   informat Capital $11. ;
   informat Bird $20. ;
   format Region $7. ;
   format State $16. ;
   format Capital $11. ;
   format Bird $20. ;
   input
      Region $ 
      State $ 
      Capital $ 
      Bird $ 
   ;
   if _ERROR_ then call symputx('_EFIERR_','1'); /* set ERROR detection macro variable */
run;
```
NOTE: The infile STDATA is:
Filename=/userid/pathname/state_data.txt,
Owner Name=userid, Group Name=unix_pubs,
Access Permission=rwx------,
Last Modified=25Mar2016:14:05:22,
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

OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;

Output Examples

Output 9.2 Work.States Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>Region</th>
<th>State</th>
<th>Capital</th>
<th>Bird</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>South</td>
<td>Georgia</td>
<td>Atlanta</td>
<td>Brown Thrasher</td>
</tr>
<tr>
<td>2</td>
<td>South</td>
<td>North Carolina</td>
<td>Raleigh</td>
<td>Cardinal</td>
</tr>
<tr>
<td>3</td>
<td>North</td>
<td>Connecticut</td>
<td>Hartford</td>
<td>Robin</td>
</tr>
<tr>
<td>4</td>
<td>West</td>
<td>Washington</td>
<td>Olympia</td>
<td>American Goldfinch</td>
</tr>
<tr>
<td>5</td>
<td>Midwest</td>
<td>Illinois</td>
<td>Springfield</td>
<td>Cardinal</td>
</tr>
</tbody>
</table>
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.

*Input Data 9.1 Input*

<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joyce</td>
<td>F</td>
<td>11</td>
</tr>
<tr>
<td>Thomas</td>
<td>M</td>
<td>11</td>
</tr>
<tr>
<td>Jane</td>
<td>F</td>
<td>12</td>
</tr>
<tr>
<td>Louise</td>
<td>F</td>
<td>12</td>
</tr>
<tr>
<td>James</td>
<td>M</td>
<td>12</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>12</td>
</tr>
<tr>
<td>Robert</td>
<td>M</td>
<td>12</td>
</tr>
<tr>
<td>Alice</td>
<td>F</td>
<td>13</td>
</tr>
<tr>
<td>Barbara</td>
<td>F</td>
<td>13</td>
</tr>
<tr>
<td>Jeffery</td>
<td>M</td>
<td>13</td>
</tr>
<tr>
<td>Carol</td>
<td>F</td>
<td>14</td>
</tr>
<tr>
<td>Judy</td>
<td>F</td>
<td>14</td>
</tr>
<tr>
<td>Alfred</td>
<td>M</td>
<td>14</td>
</tr>
<tr>
<td>Henry</td>
<td>M</td>
<td>14</td>
</tr>
<tr>
<td>Jenet</td>
<td>F</td>
<td>15</td>
</tr>
<tr>
<td>Mary</td>
<td>F</td>
<td>15</td>
</tr>
<tr>
<td>Ronald</td>
<td>M</td>
<td>15</td>
</tr>
<tr>
<td>William</td>
<td>M</td>
<td>15</td>
</tr>
<tr>
<td>Philip</td>
<td>M</td>
<td>16</td>
</tr>
</tbody>
</table>

**Program**

```sas
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.
Example 3: Importing a Tab-Delimited File

Log 9.3  Importing a Tab-Delimited File

```sas
OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
proc import datafile='/userid/pathname/Class.txt' out=class dbms=dlm replace;
datarow=5;
delimiter='09'x;
run;

/***************************************************************/
* PRODUCT:   SAS
* VERSION:   V.03.00
* CREATOR:   External File Interface
* DATE:      25MAR16
* DESC:      Generated SAS Datastep Code
* TEMPLATE SOURCE:  (None Specified.)
/***************************************************************/
data WORK.CLASS    ;
%let _EFIERR_ = 0; /* set the ERROR detection macro variable */
infile '/userid/pathname/Class.txt' delimiter='09'x MISSOVER DSD lrecl=32767
firstobs=5 ;
informat "Name     Gender   Age"N $20. ;
format "Name     Gender   Age"N $20. ;
input
"Name     Gender   Age"N $
;
if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection macro variable */
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=25Mar2016:15:39:54,
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
proc print data=class;
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
OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
```
Output Examples

Output 9.3  Work.Class Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Gender</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Louise</td>
<td>F</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>James</td>
<td>M</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>M</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Robert</td>
<td>M</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>Alice</td>
<td>F</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Barbara</td>
<td>F</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>Jeffery</td>
<td>M</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>Carol</td>
<td>F</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>Judy</td>
<td>F</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>Alfred</td>
<td>M</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>Henry</td>
<td>M</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>Janet</td>
<td>F</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>Mary</td>
<td>F</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>Ronald</td>
<td>M</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>William</td>
<td>M</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>Philip</td>
<td>M</td>
<td>16</td>
</tr>
</tbody>
</table>

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.

"Asia", "Bangkok", 1", "$9,996", "$9,576", "$80"
"Canada", "Calgary", 8", "$17,720", "$63,280", "$472"
"Central America/Caribbean", "Kingston", 33", "$102,372", "$393,376", "$4,454"
"Eastern Europe", "Budapest", 22", "$74,102", "$317,515", "$3,341"
"Middle East", "Al-Khobar", 10", "$15,062", "$44,658", "$765"
"Pacific", "Auckland", 12", "$20,141", "$97,919", "$962"
"South America", "Bogota", 19", "$15,312", "$35,805", "$1,229"
"United States", "Chicago", 16", "$82,483", "$305,061", "$3,735"
"Western Europe", "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.
importing a comma-delimited file

```
proc import datafile="/userid/pathname/shoes.txt" out=shoes dbms=csv replace;
getnames=no;
run;

/*********************************************************
* PRODUCT:    SAS
* VERSION:    V.03.00
* CREATOR:    External File Interface
* DATE:       25MAR16
* DESC:       Generated SAS Datastep Code
* TEMPLATE SOURCE: (None Specified.)
***********************************************************/
data WORK.SHOES ;
%let _EFIERR_ = 0; /* set the ERROR detection macro variable */
infile '/userid/pathname/shoes.txt' delimiter = ',' MISSOVER DSD lrecl=32767 ;
informat VAR1 $27. ;
informat VAR2 $6. ;
informat VAR3 $13. ;
informat VAR4 $4. ;
informat VAR5 $10. ;
informat VAR6 $10. ;
informat VAR7 $8. ;
format VAR1 $27. ;
format VAR2 $6. ;
format VAR3 $13. ;
format VAR4 $4. ;
format VAR5 $10. ;
format VAR6 $10. ;
format VAR7 $8. ;
input VAR1 $
          VAR2 $
          VAR3 $
          VAR4 $
          VAR5 $
          VAR6 $
          VAR7 $
;  
if _ERROR_ then call symputx('_EFIERR_','1'); /* set ERROR detection macro variable */
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=25Mar2016:15:52:41,
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

proc print data=work.shoes;
run;

NOTE: There were 10 observations read from the data set WORK.SHOES.
NOTE: PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used (Total process time):
real time 0.07 seconds
cpu time 0.04 seconds

OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;

Output Examples

Output 9.4  Work.Shoes Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR3</th>
<th>VAR4</th>
<th>VAR5</th>
<th>VAR6</th>
<th>VAR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Africa</td>
<td>Boot</td>
<td>Addis Ababa</td>
<td>12</td>
<td>$29,761</td>
<td>$191,821</td>
<td>$759</td>
</tr>
<tr>
<td>2</td>
<td>Asia</td>
<td>Boot</td>
<td>Bangkok</td>
<td>1</td>
<td>$1,996</td>
<td>$9,576</td>
<td>$80</td>
</tr>
<tr>
<td>3</td>
<td>Canada</td>
<td>Boot</td>
<td>Calgary</td>
<td>8</td>
<td>$17,720</td>
<td>$63,280</td>
<td>$472</td>
</tr>
<tr>
<td>4</td>
<td>Central America/Caribbean</td>
<td>Boot</td>
<td>Kingston</td>
<td>33</td>
<td>$102,372</td>
<td>$393,376</td>
<td>$4,454</td>
</tr>
<tr>
<td>5</td>
<td>Eastern Europe</td>
<td>Boot</td>
<td>Budapest</td>
<td>22</td>
<td>$74,102</td>
<td>$317,515</td>
<td>$3,341</td>
</tr>
<tr>
<td>6</td>
<td>Middle East</td>
<td>Boot</td>
<td>Al-Khobar</td>
<td>10</td>
<td>$15,062</td>
<td>$44,658</td>
<td>$755</td>
</tr>
<tr>
<td>7</td>
<td>Pacific</td>
<td>Boot</td>
<td>Auckland</td>
<td>12</td>
<td>$20,141</td>
<td>$97,919</td>
<td>$962</td>
</tr>
<tr>
<td>8</td>
<td>South America</td>
<td>Boot</td>
<td>Bogota</td>
<td>19</td>
<td>$15,312</td>
<td>$35,605</td>
<td>$1,229</td>
</tr>
<tr>
<td>9</td>
<td>United States</td>
<td>Boot</td>
<td>Chicago</td>
<td>16</td>
<td>$82,463</td>
<td>$305,061</td>
<td>$3,735</td>
</tr>
<tr>
<td>10</td>
<td>Western Europe</td>
<td>Boot</td>
<td>Copenhagen</td>
<td>2</td>
<td>$1,663</td>
<td>$4,657</td>
<td>$129</td>
</tr>
</tbody>
</table>
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)>;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC JAVAINFO</td>
<td>Display diagnostic information about the SAS Java environment</td>
</tr>
</tbody>
</table>

PROC JAVAINFO Statement

Displays diagnostic information about the SAS Java environment.

Syntax

PROC JAVAINFO <option(s)>;
Optional Arguments

ALL
  specifies current information about the SAS Java environment.

CLASSPATHS
  specifies information about the classpaths that Java is using.

HELP
  specifies usage assistance in using the JAVAINFO procedure.

JREOPTIONS
  specifies the Java properties that are set when the JREOPTIONS configuration option is specified.
  • When used in PROC JAVAINFO, JREOPTIONS specifies the JREOPTIONS Java properties that are set when Java is started.
  • When used in PROC OPTIONS, JREOPTIONS specifies the Java options that are in the configuration file when SAS is started.

Note: SAS.cfg is the configuration file specified during installation, but other configuration files can be specified.

OS
  specifies information about the operating system that SAS is running under.

version
  specifies the Java Runtime Environment (JRE) that SAS is using.
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)>;**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC OPTIONS</td>
<td>List the current system option settings to the SAS Log</td>
<td>Ex. 1, Ex. 2, Ex. 3, Ex. 4</td>
</tr>
</tbody>
</table>

### 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 230
“Example 2: Displaying the Setting of a Single Option” on page 231
“Example 3: Displaying Expanded Path Environment Variables” on page 232
“Example 4: List the Options That Can Be Specified by the INSERT and APPEND Options” on page 234

### 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 231

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 223 option to view paths that display the environment variable

Example “Example 3: Displaying Expanded Path Environment Variables” on page 232

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 227

HEXVALUE
displays system option character values as hexadecimal values.

HOST
displays only host options.

See “NOHOST” on page 223 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 LISTINSERTAPPEND 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 234

LISTGROUPS
lists the system option groups as well as a description of each group.

See “Displaying Information about System Option Groups” on page 227

LISTRESTRICT
lists the system options that can be restricted by your site administrator.

See “RESTRICT” on page 223 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 224 option to produce a compressed listing without descriptions

Example “Example 1: Producing the Short Form of the Options Listing” on page 230

LOGNUMBERFORMAT
displays numeric system option values using locale-specific punctuation.

See “NOLOGNUMBERFORMAT” on page 223 option to display numeric option values without using commas

Example “Example 2: Displaying the Setting of a Single Option” on page 231

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 221 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 222 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 223 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 231

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 222 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 223 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 EMAILPW, return the value xxxxxxxx and not the actual password.

Example “Example 2: Displaying the Setting of a Single Option” on page 231

---

**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.

```sas
proc options;
run;
```
Log 11.1  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 11.2  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 11.3  The Setting of a Single SAS System Option

```sas
56   proc options option=errorcheck define; run;
```

SAS (r) Proprietary Software Release V.03.00  TSIM0

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:

```sas
proc options option=(append insert) define; run;
```

Log 11.4  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 11.5  List of SAS System Option Groups

    56       proc options listgroups; run;

SAS (r) Proprietary Software Release V.03.00  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
Use the GROUP= option to display system options that belong to a particular group. You can specify one or more groups.

```sas
proc options group=(cas memory);
run;
```
Log 11.6  Sample Output Using the GROUP= Option

56         proc options group=(cas memory); run;

SAS (r) Proprietary Software Release V.03.00  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.
CASWORKERS=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 11.7 A List of Options That Have Been Restricted by the Site Administrator

```
proc options restrict;
run;
```

Option Value Information For SAS Option EMAILSYS
Value: SMPT
Scope: SAS Session
How option value set: Site Administrator Restricted

Log 11.8 A Partial Log That Lists Options That Can Be Restricted

```
proc options listrestrict; run;
```

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` | `NOoption` are listed as either `option` or `NOoption`, 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 224.

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 11.9  Partial Listing of the SHORT Option

56 proc options short; run;

SAS (r) Proprietary Software Release V.03.00  TS1M0

Portable Options:
APPEND= APPLETOC=/pathname  AUTOCORRECT AUTOEXEC=/opt/sas/viya/etc/workspaceserver/autoexec.sas
BINDING=DEFAULT BOTTMARGIN=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 11.10  Log Output from Specifying the MEMSIZE Option

```
56         proc options option=memsize define value lognumberformat;
57         run;

SAS (r) Proprietary Software Release V.03.00  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

```sas
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`

```sas
proc options option=msg expand;
run;
proc options option=msg noexpand;
run;
```

Log

**Log 11.11  Displaying an Expanded and Nonexpanded Pathname Using the OPTIONS Procedure**

```sas
56         proc options option=msg expand;
57         run;

SAS (r) Proprietary Software Release V.03.00  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.03.00  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 11.12 Displaying the Options That Can Be Specified by the INSERT and APPEND Options

56 proc options listinsertappend; run;

SAS (r) Proprietary Software Release V.03.00 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.
SASHHELP 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 12
PRINT Procedure

Overview: PRINT Procedure ........................................ 236
   What Does the PRINT Procedure Do? .......................... 236
   A Simple Report ................................................. 236
   Customized Report ............................................. 237

Concepts: PRINT Procedure ...................................... 238
   About PROC PRINT Output .................................... 238
   Page Layout for HTML5, the Default ODS Destination ...... 239
   Page Layout for PDF and RTF Destinations with Limited Page Sizes ... 239

Syntax: PRINT Procedure ........................................ 241
   PROC PRINT Statement ........................................ 242
   BY Statement .................................................. 252
   ID Statement .................................................. 253
   PAGEBY Statement ............................................ 255
   SUM Statement ................................................ 255
   SUMBY Statement .............................................. 256
   VAR Statement ................................................ 257

Printing CAS Tables ................................................ 258

Error Processing in the PRINT Procedure Output .............. 258

Examples: PRINT Procedure ....................................... 258
   Example 1: Print a CAS Table ................................ 258
   Example 2: Selecting Variables to Print ..................... 260
   Example 3: Customizing Text in Column Headings Using a Batch Session ... 263
   Example 4: Creating Separate Sections of a Report for
              Groups of Rows Using a Batch Session .................. 267
   Example 5: Summing Numeric Variables with One BY Group ................ 274
   Example 6: Summing Numeric Variables with Multiple BY Variables ...... 277
   Example 7: Limiting the Number of Sums in a Report .............. 282
   Example 8: Creating a Customized Layout with BY Groups and ID Variables ... 286
   Example 9: Printing All the Data Sets in a SAS Library .......... 291
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 260 creates the data set EXPREV.

```sas
options obs=10;
proc print data=exprev;
run;
```

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>Country</th>
<th>Emp_ID</th>
<th>Order_Date</th>
<th>Ship_Date</th>
<th>Sale_Type</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Antarktica</td>
<td>999999999</td>
<td>1/1/16</td>
<td>1/7/16</td>
<td>Internet</td>
<td>2</td>
<td>92.6</td>
<td>20.70</td>
</tr>
<tr>
<td>2</td>
<td>Puerto Rico</td>
<td>999999999</td>
<td>1/1/16</td>
<td>1/5/16</td>
<td>Catalog</td>
<td>14</td>
<td>51.2</td>
<td>12.10</td>
</tr>
<tr>
<td>3</td>
<td>Virgin Islands</td>
<td>999999999</td>
<td>1/1/16</td>
<td>1/4/16</td>
<td>In Store</td>
<td>25</td>
<td>31.1</td>
<td>15.65</td>
</tr>
<tr>
<td>4</td>
<td>Aruba</td>
<td>999999999</td>
<td>1/1/16</td>
<td>1/4/16</td>
<td>Catalog</td>
<td>30</td>
<td>123.7</td>
<td>59.00</td>
</tr>
<tr>
<td>5</td>
<td>Bahamas</td>
<td>999999999</td>
<td>1/1/16</td>
<td>1/4/16</td>
<td>Catalog</td>
<td>8</td>
<td>113.4</td>
<td>28.45</td>
</tr>
<tr>
<td>6</td>
<td>Bermuda</td>
<td>999999999</td>
<td>1/1/16</td>
<td>1/4/16</td>
<td>Catalog</td>
<td>7</td>
<td>41.0</td>
<td>9.25</td>
</tr>
<tr>
<td>7</td>
<td>Belize</td>
<td>120458</td>
<td>1/2/16</td>
<td>1/2/16</td>
<td>In Store</td>
<td>2</td>
<td>146.4</td>
<td>36.70</td>
</tr>
<tr>
<td>8</td>
<td>British Virgin Islands</td>
<td>999999999</td>
<td>1/2/16</td>
<td>1/5/16</td>
<td>Catalog</td>
<td>11</td>
<td>40.2</td>
<td>20.20</td>
</tr>
<tr>
<td>9</td>
<td>Canada</td>
<td>999999999</td>
<td>1/2/16</td>
<td>1/5/16</td>
<td>Catalog</td>
<td>100</td>
<td>11.8</td>
<td>5.00</td>
</tr>
<tr>
<td>10</td>
<td>Cayman Islands</td>
<td>120454</td>
<td>1/2/16</td>
<td>1/2/16</td>
<td>In Store</td>
<td>20</td>
<td>71.0</td>
<td>32.30</td>
</tr>
</tbody>
</table>

This next example creates the CAS table Mycas.Cars as a subset of the Sashelp.cars data set:

```sas
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;

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 289.
Figure 12.1 Customized Report Produced by PROC PRINT

Expenses Incurred for Salaries for Flight Attendants and Mechanics

<table>
<thead>
<tr>
<th>Job Code</th>
<th>Gender</th>
<th>Annual Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA1</td>
<td>F</td>
<td>$23,177.00</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>$22,454.00</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>$22,268.00</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>$67,899.00</td>
</tr>
<tr>
<td>FA2</td>
<td>F</td>
<td>$28,888.00</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>$27,787.00</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>$28,572.00</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>$85,247.00</td>
</tr>
<tr>
<td>FA3</td>
<td>F</td>
<td>$32,886.00</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>$33,419.00</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>$32,217.00</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>$98,522.00</td>
</tr>
<tr>
<td>ME1</td>
<td>M</td>
<td>$29,769.00</td>
</tr>
<tr>
<td>Grand Total</td>
<td></td>
<td>$281,437.00</td>
</tr>
</tbody>
</table>

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 260 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 12.2 Printing Rows on a Single Line

<table>
<thead>
<tr>
<th>Obs</th>
<th>Var_1</th>
<th>Var_2</th>
<th>Var_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>3</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>4</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>5</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>6</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>

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.
## Column Headings

The amount of spacing specifies whether PROC PRINT prints column headings horizontally or vertically. Figure 12.2 on page 239, Figure 12.3 on page 240, and Figure 12.4 on page 240 all illustrate horizontal headings. The following figure illustrates vertical headings.

### Figure 12.3 Splitting Rows into Multiple Sections on One Page

<table>
<thead>
<tr>
<th>Obs</th>
<th>Var_1</th>
<th>Var_2</th>
<th>Var_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>3</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>Var_4</th>
<th>Var_5</th>
<th>Var_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>3</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>

### Figure 12.4 Splitting Rows across Multiple Pages

<table>
<thead>
<tr>
<th>Obs</th>
<th>Var_1</th>
<th>Var_2</th>
<th>Var_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>3</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>Var_4</th>
<th>Var_5</th>
<th>Var_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>3</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>Var_7</th>
<th>Var_8</th>
<th>Var_9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>3</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>Var_10</th>
<th>Var_11</th>
<th>Var_12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>3</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>
Figure 12.5 Using Vertical Headings

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)
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-attribute-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-attribute-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 247 for valid style attributes.

Example “Example 2: Selecting Variables to Print” on page 260

**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 277

**HEADING=direction**

does 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.

**Aliases**  
H  
V

**Default**  
Heads 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.

**Aliases**  
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 267 |

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.

<table>
<thead>
<tr>
<th>N Option Use</th>
<th>PROC PRINT Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>With neither a BY nor a SUM statement</td>
<td>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.</td>
</tr>
<tr>
<td>With a BY statement</td>
<td>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.</td>
</tr>
<tr>
<td>With a BY statement and a SUM statement</td>
<td>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.</td>
</tr>
</tbody>
</table>

| Examples | “Example 3: Customizing Text in Column Headings Using a Batch Session” on page 263 |
| Examples | “Example 4: Creating Separate Sections of a Report for Groups of Rows Using a Batch Session” on page 267 |
| Examples | “Example 5: Summing Numeric Variables with One BY Group” on page 274 |
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 267

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 246.)

Example
“Example 3: Customizing Text in Column Headings Using a Batch Session” on page 263

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 246.)

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 263

**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>
<thead>
<tr>
<th>Location</th>
<th>Location Alias</th>
<th>Affected Report Part</th>
<th>Can Also Be Used in These Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYLABEL</td>
<td>BYSUMLABEL</td>
<td>Label for the BY variable on the line containing the SUM totals</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>BYLBL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BYSUMLBL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>COLUMN</td>
<td>All data except for data in the OBS column or the ID columns</td>
<td>VAR</td>
</tr>
<tr>
<td></td>
<td>COL</td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or</td>
<td>SUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data in the ID columns when the DATA location is specified in the STYLE= option of the ID statement</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>Location Alias</td>
<td>Affected Report Part</td>
<td>Can Also Be Used in These Statements</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>GRANDTOTAL</td>
<td>GRANDTOT</td>
<td>SUM line containing the grand totals for the whole report</td>
<td>SUM</td>
</tr>
<tr>
<td></td>
<td>GRAND</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GTOTAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GTOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEADER</td>
<td>HEAD</td>
<td>All column headings except for the OBS column or the ID columns</td>
<td>VAR</td>
</tr>
<tr>
<td></td>
<td>HDR</td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or</td>
<td>SUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All column headings of the ID columns when the HEADER location is specified</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>in the STYLE= option of the ID statement</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>None</td>
<td>N= table and contents</td>
<td>None</td>
</tr>
<tr>
<td>OBS</td>
<td>OBSDATA</td>
<td>Data in the OBS column or the ID columns unless the DATA location is specified in the</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>OBSCOLUMN</td>
<td>STYLE= option of the ID statement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OBSCOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBSHEADER</td>
<td>OBSHEAD</td>
<td>Header of the OBS column or the ID columns unless the HEADER location is specified</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>OBSHDR</td>
<td>in the STYLE= option of the ID statement</td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td>REPORT</td>
<td>Structural part of the report - that is, the underlying table used to set things</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>like the width of the border and the space between cells</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>TOT</td>
<td>SUM line containing totals for each BY group</td>
<td>SUM</td>
</tr>
<tr>
<td></td>
<td>BYSUMLINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BYLINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BYSUM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 12.6  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:

```sql
proc print data=mycas.exprev style(head)={backgroundcolor=black color=white};
id country / style(header data)=[backgroundcolor=gray];
run;
```

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]`.
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;
```

---

### 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 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKGROUNDCOLOR</td>
<td></td>
</tr>
<tr>
<td>FONTWIDTH</td>
<td>*</td>
</tr>
<tr>
<td>BACKGROUNDIMAGE</td>
<td></td>
</tr>
<tr>
<td>COLOR</td>
<td>*</td>
</tr>
<tr>
<td>BORDERCOLOR</td>
<td></td>
</tr>
<tr>
<td>FRAME</td>
<td></td>
</tr>
<tr>
<td>BORDERCOLORDARK</td>
<td></td>
</tr>
<tr>
<td>HTMLCLASS</td>
<td></td>
</tr>
<tr>
<td>BORDERCOLORLIGHT=</td>
<td>TEXTALIGN=</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>BORDERWIDTH=</td>
<td>OUTPUTWIDTH=</td>
</tr>
<tr>
<td>CELLPADDING=</td>
<td>POSTHTML=</td>
</tr>
<tr>
<td>CELLSspacing=</td>
<td>POSTIMAGE=</td>
</tr>
<tr>
<td>FONT=*</td>
<td>POSTTEXT=</td>
</tr>
<tr>
<td>FONTFAMILY=*</td>
<td>PREHTML=</td>
</tr>
<tr>
<td>FONTsize=*</td>
<td>PREIMAGE=</td>
</tr>
<tr>
<td>FONTstyle=*</td>
<td>PRETEXT=</td>
</tr>
<tr>
<td>FONTweight=*</td>
<td>RULES=</td>
</tr>
</tbody>
</table>

* 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:

<table>
<thead>
<tr>
<th>ASIS=</th>
<th>FONTWIDTH=</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKGROUNDCOLOR=</td>
<td>HREFTARGET=</td>
</tr>
<tr>
<td>BACKGROUNDIMAGE=</td>
<td>CLASS=</td>
</tr>
<tr>
<td>BORDERCOLOR=</td>
<td>TEXTALIGN=</td>
</tr>
<tr>
<td>BORDERCOLORDARK=</td>
<td>NOBREAKSPACE=</td>
</tr>
<tr>
<td>BORDERCOLORLIGHT=</td>
<td>POSTHTML=</td>
</tr>
<tr>
<td>BORDERWIDTH=</td>
<td>POSTIMAGE=</td>
</tr>
<tr>
<td>HEIGHT=</td>
<td>POSTTEXT=</td>
</tr>
<tr>
<td>CELLWIDTH=</td>
<td>PREHTML=</td>
</tr>
<tr>
<td>FLYOVER=</td>
<td>PREIMAGE=</td>
</tr>
<tr>
<td>FONT=</td>
<td>PRETEXT=</td>
</tr>
<tr>
<td>FONTFAMILY=</td>
<td>PROTECTSPECIALCHARACTERS=</td>
</tr>
<tr>
<td>FONTsize=</td>
<td>TAGATTR=</td>
</tr>
<tr>
<td>FONTstyle=</td>
<td>URL=</td>
</tr>
</tbody>
</table>
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 267
“Example 5: Summing Numeric Variables with One BY Group” on page 274
“Example 6: Summing Numeric Variables with Multiple BY Variables” on page 277
“Example 7: Limiting the Number of Sums in a Report” on page 282
“Example 8: Creating a Customized Layout with BY Groups and ID Variables” on page 286

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 286.)

**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 286

---

**Syntax**

```plaintext
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:

```
sty le-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 247 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 286.)
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 267

**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 274

"Example 6: Summing Numeric Variables with Multiple BY Variables" on page 277

"Example 7: Limiting the Number of Sums in a Report" on page 282

"Example 8: Creating a Customized Layout with BY Groups and ID Variables" on page 286

**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 243 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 274.)

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 277.)

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 282
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.

Syntax

VAR variable(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 247 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

<table>
<thead>
<tr>
<th>Features:</th>
<th>PROC PRINT DATA=CAS-table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other features:</td>
<td>CAS language elements</td>
</tr>
<tr>
<td></td>
<td>CAS statement</td>
</tr>
<tr>
<td></td>
<td>LIBNAME statement for the CAS engine</td>
</tr>
<tr>
<td></td>
<td>PROC CASUTIL</td>
</tr>
<tr>
<td></td>
<td>PROC MDSUMMARY</td>
</tr>
<tr>
<td>Data set:</td>
<td>Sashelp.cars</td>
</tr>
</tbody>
</table>
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

```sas
  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.

```sas
  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.

```sas
  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.

```sas
  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_.

```plaintext
options obs=15;
proc print data=mycas.mpghw_sum;
    var origin type _mean_;
run;
```

![Average Highway Milages Table](image)

**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**

```plaintext
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.

```plaintext
options obs=10;
```

**Print the output** The VAR statement specifies the variables to print.

```plaintext
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 12.1  Selecting Variables: Default HTML5 Output**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Country</th>
<th>Price</th>
<th>Sale_Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Antarctica</td>
<td>92.6</td>
<td>Internet</td>
</tr>
<tr>
<td>2</td>
<td>Puerto Rico</td>
<td>51.2</td>
<td>Catalog</td>
</tr>
<tr>
<td>3</td>
<td>Virgin Islands (U.S.)</td>
<td>31.1</td>
<td>In Store</td>
</tr>
<tr>
<td>4</td>
<td>Aruba</td>
<td>123.7</td>
<td>Catalog</td>
</tr>
<tr>
<td>5</td>
<td>Bahamas</td>
<td>113.4</td>
<td>Catalog</td>
</tr>
<tr>
<td>6</td>
<td>Bermuda</td>
<td>41.0</td>
<td>Catalog</td>
</tr>
<tr>
<td>7</td>
<td>Belize</td>
<td>146.4</td>
<td>In Store</td>
</tr>
<tr>
<td>8</td>
<td>British Virgin Islands</td>
<td>40.2</td>
<td>Catalog</td>
</tr>
<tr>
<td>9</td>
<td>Canada</td>
<td>11.8</td>
<td>Catalog</td>
</tr>
<tr>
<td>10</td>
<td>Cayman Islands</td>
<td>71.0</td>
<td>In Store</td>
</tr>
</tbody>
</table>

*prices in USD
Program: Creating an HTML5 Report with the STYLE and BLANKLINE Options

```sas
options obs=5;

proc print data=exprev
  style(header)={fontstyle=italic color= green}
  style(obs)={backgroundcolor=a8a4ff8a 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.

```sas
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 a8a4ff8a 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.

```sas
proc print data=exprev
  style(header)={fontstyle=italic color= green}
  style(obs)={backgroundcolor=a8a4ff8a color=blue}
  blankline=(count= 1 style={backgroundcolor=cx456789});
var country sale_type;
  title 'Monthly Price Per Unit and Sale Type for Each Country';
  footnote '*prices in USD';
run;
```
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

---

Output: HTML5 Output with Styles

*Output 12.2  Selecting Variables: HTML5 Output Using Styles*

**Monthly Price Per Unit and Sale Type for Each Country**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Country</th>
<th>Price</th>
<th>Sale_Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Antarctica</td>
<td>92.6</td>
<td>Internet</td>
</tr>
<tr>
<td>2</td>
<td>Puerto Rico</td>
<td>51.2</td>
<td>Catalog</td>
</tr>
<tr>
<td>3</td>
<td>Virgin Islands (U.S.)</td>
<td>31.1</td>
<td>In Store</td>
</tr>
<tr>
<td>4</td>
<td>Aruba</td>
<td>123.7</td>
<td>Catalog</td>
</tr>
<tr>
<td>5</td>
<td>Bahamas</td>
<td>113.4</td>
<td>Catalog</td>
</tr>
</tbody>
</table>

*prices in USD*
• 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

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
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.

```plaintext
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 12.3 Customizing Column Heading: Default PDF Output**

### Program: Creating a PDF Report with the STYLE Option, in Batch

```options obs=10 nodate;
ods pdf file='your_file.pdf';
proc print data=experv 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 create 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;
  ```

- **Close the PDF destination.** The ODS PDF CLOSE statement closes the PDF destination.
  
  ```
  ods pdf close;
  ```
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 demonstrate 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

```sas
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.

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.

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.
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;
**Creating Separate Sections of a Report for Groups of Rows: Default RTF Output**

Price and Cost Grouped by Date and Order Type in USD

<table>
<thead>
<tr>
<th>Order Type=Catalog Order Date=1/1/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>30</td>
</tr>
</tbody>
</table>
Number of observations for each order type: 4

<table>
<thead>
<tr>
<th>Order Type=Catalog Order Date=1/2/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>
Number of observations for each order type: 2

<table>
<thead>
<tr>
<th>Order Type=In Store Order Date=1/1/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
</tr>
<tr>
<td>25</td>
</tr>
</tbody>
</table>
Number of observations for each order type: 1

<table>
<thead>
<tr>
<th>Order Type=In Store Order Date=1/2/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>
Number of observations for each order type: 2

<table>
<thead>
<tr>
<th>Order Type=Internet Order Date=1/1/16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
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;
   var quantity / style(header)=[backgroundcolor=very light gray];
   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 Expprev 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 Expprev 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.experv 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;
```
### Prices and Cost Grouped by Date and Order Type
*prices in USD*

**Order Type=Catalog Order Date=1/1/16**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Cost</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>$12.10</td>
<td>$51.20</td>
</tr>
<tr>
<td>30</td>
<td>$59.00</td>
<td>$123.70</td>
</tr>
<tr>
<td>8</td>
<td>$28.45</td>
<td>$113.40</td>
</tr>
<tr>
<td>7</td>
<td>$9.25</td>
<td>$41.00</td>
</tr>
</tbody>
</table>

Number of observations for the month: 4

**Order Type=Catalog Order Date=1/2/16**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Cost</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>$5.00</td>
<td>$11.80</td>
</tr>
<tr>
<td>11</td>
<td>$20.20</td>
<td>$40.20</td>
</tr>
</tbody>
</table>

Number of observations for the month: 2

**Order Type=In Store Order Date=1/1/16**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Cost</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>$15.85</td>
<td>$31.10</td>
</tr>
</tbody>
</table>

Number of observations for the month: 1

**Order Type=In Store Order Date=1/2/16**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Cost</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>$32.30</td>
<td>$71.00</td>
</tr>
<tr>
<td>2</td>
<td>$36.70</td>
<td>$146.40</td>
</tr>
</tbody>
</table>

Number of observations for the month: 2

**Order Type=Internet Order Date=1/1/16**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Cost</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$20.70</td>
<td>$92.60</td>
</tr>
</tbody>
</table>

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

```sas
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;
```
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.

Sort the data set. PROC SORT sorts the rows by Sale_Type.

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.

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.

Select the variables to include in the report. The VAR statement creates columns for Country, Order_Date, Quantity, and Price, in that order.

Assign the variables’ labels as column headings. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step.
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.

```sas
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.

```sas
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.

```sas
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.

```sas
options byline;
```

---

**Output: HTML5**

*Output 12.7 Summing Numeric Variables with One BY Group HTML5 Output*
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;
### Retail and Quantity Totals for Each Sale Date and Sale Type

#### Sale Type=Catalog  Sale Date=1/1/16

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puerto Rico</td>
<td>999999999</td>
<td>1/5/16</td>
<td>14</td>
<td>$51.20</td>
<td>$12.10</td>
</tr>
<tr>
<td>Aruba</td>
<td>999999999</td>
<td>1/4/16</td>
<td>30</td>
<td>$123.70</td>
<td>$59.00</td>
</tr>
<tr>
<td>Bahamas</td>
<td>999999999</td>
<td>1/4/16</td>
<td>8</td>
<td>$113.40</td>
<td>$28.45</td>
</tr>
<tr>
<td>Bermuda</td>
<td>999999999</td>
<td>1/4/16</td>
<td>7</td>
<td>$41.00</td>
<td>$9.25</td>
</tr>
<tr>
<td><strong>Totals</strong></td>
<td></td>
<td></td>
<td><strong>59</strong></td>
<td><strong>$329.30</strong></td>
<td><strong>$108.80</strong></td>
</tr>
</tbody>
</table>

N = 4

#### Sale Type=Catalog  Sale Date=1/2/16

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>British Virgin Islands</td>
<td>999999999</td>
<td>1/5/16</td>
<td>11</td>
<td>$40.20</td>
<td>$20.20</td>
</tr>
<tr>
<td>Canada</td>
<td>999999999</td>
<td>1/5/16</td>
<td>100</td>
<td>$11.80</td>
<td>$5.00</td>
</tr>
<tr>
<td><strong>Totals</strong></td>
<td></td>
<td></td>
<td><strong>111</strong></td>
<td><strong>$52.00</strong></td>
<td><strong>$25.20</strong></td>
</tr>
</tbody>
</table>

**Totals**

N = 2
**Program: Creating an HTML5 Report with the STYLE Option**

```plaintext
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**

```plaintext
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.

```plaintext
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 12.9  Summing Numeric Variables with Multiple BY Variables: Catalog Sales: HTML5 Output Using Styles
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

```plaintext
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.**

```r
options obs=10;
```

**Sort the data set.** PROC SORT sorts the rows by Sales_Type and Order_Date.

```r
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’.

```r
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.

```r
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.

```r
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.

```r
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 12.10** 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**

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puerto Rico</td>
<td>9999999999</td>
<td>1/5/12</td>
<td>14</td>
<td>$51.20</td>
<td>$12.10</td>
</tr>
<tr>
<td>Araba</td>
<td>9999999999</td>
<td>1/4/12</td>
<td>30</td>
<td>$123.70</td>
<td>$59.00</td>
</tr>
<tr>
<td>Bahamas</td>
<td>9999999999</td>
<td>1/4/12</td>
<td>8</td>
<td>$113.40</td>
<td>$28.45</td>
</tr>
<tr>
<td>Beninada</td>
<td>9999999999</td>
<td>1/4/12</td>
<td>7</td>
<td>$41.00</td>
<td>$9.25</td>
</tr>
</tbody>
</table>

**Sale Type=Catalog Sale Date=1/2/12**

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>British Virgin Islands</td>
<td>9999999999</td>
<td>1/5/12</td>
<td>11</td>
<td>$40.20</td>
<td>$20.20</td>
</tr>
<tr>
<td>Canada</td>
<td>9999999999</td>
<td>1/5/12</td>
<td>100</td>
<td>$11.90</td>
<td>$5.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>170</td>
<td>$381.30</td>
<td></td>
</tr>
</tbody>
</table>

### Sale Type=In Store Sale Date=1/1/12

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virgin Islands (U.S.)</td>
<td>9999999999</td>
<td>1/4/12</td>
<td>25</td>
<td>$31.15</td>
<td>$15.65</td>
</tr>
</tbody>
</table>

### Sale Type=In Store Sale Date=1/2/12

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belize</td>
<td>120458</td>
<td>1/2/12</td>
<td>2</td>
<td>$186.40</td>
<td>$36.70</td>
</tr>
<tr>
<td>Cayman Islands</td>
<td>120454</td>
<td>1/2/12</td>
<td>20</td>
<td>$71.00</td>
<td>$32.30</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>47</td>
<td>$248.50</td>
<td></td>
</tr>
</tbody>
</table>

### Sale Type=Internet Sale Date=1/1/12

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antarctica</td>
<td>9999999999</td>
<td>1/7/12</td>
<td>2</td>
<td>$92.60</td>
<td>$20.70</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td></td>
<td></td>
<td>219</td>
<td>$722.40</td>
<td></td>
</tr>
</tbody>
</table>

**Program: Creating a PDF Report with the STYLE Option**

```plaintext
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;
```


**Example 7: Limiting the Number of Sums in a Report**

```latex
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**

```latex
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;
run;
ods pdf close;
```

---

**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.

```latex
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 12.11  Limiting the Number of Sums in a Report: PostScript Output Using Styles

### Retail and Quantity Totals for Each Sale Type

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puerto Rico</td>
<td>9999999999</td>
<td>1/5/12</td>
<td>14</td>
<td>$51.20</td>
<td>$12.10</td>
</tr>
<tr>
<td>Aruba</td>
<td>9999999999</td>
<td>1/4/12</td>
<td>30</td>
<td>$123.70</td>
<td>$39.00</td>
</tr>
<tr>
<td>Bahamas</td>
<td>9999999999</td>
<td>1/4/12</td>
<td>8</td>
<td>$113.40</td>
<td>$28.45</td>
</tr>
<tr>
<td>Bermuda</td>
<td>9999999999</td>
<td>1/4/12</td>
<td>7</td>
<td>$41.00</td>
<td>$9.25</td>
</tr>
</tbody>
</table>

### Sale Type—Catalog Sale Date=1/2/12

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>British Virgin Islands</td>
<td>9999999999</td>
<td>1/5/12</td>
<td>11</td>
<td>$40.20</td>
<td>$30.20</td>
</tr>
<tr>
<td>Canada</td>
<td>9999999999</td>
<td>1/8/12</td>
<td>100</td>
<td>$11.80</td>
<td>$5.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>170</strong></td>
<td><strong>$381.30</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Sale Type—in Store Sale Date=1/1/12

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virgin Islands (U.S.)</td>
<td>9999999999</td>
<td>1/4/12</td>
<td>25</td>
<td>$31.10</td>
<td>$15.65</td>
</tr>
</tbody>
</table>

### Sale Type—in Store Sale Date=1/2/12

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belize</td>
<td>120458</td>
<td>1/2/12</td>
<td>2</td>
<td>$1.46</td>
<td>$0.70</td>
</tr>
<tr>
<td>Cayman Islands</td>
<td>120454</td>
<td>1/2/12</td>
<td>20</td>
<td>$71.00</td>
<td>$32.30</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>47</strong></td>
<td><strong>$248.50</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Sale Type—Internet Sale Date=1/1/12

<table>
<thead>
<tr>
<th>Country</th>
<th>Emp_ID</th>
<th>Ship_Date</th>
<th>Quantity</th>
<th>Price</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antartica</td>
<td>9999999999</td>
<td>1/7/12</td>
<td>2</td>
<td>$92.60</td>
<td>$20.70</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td></td>
<td></td>
<td><strong>219</strong></td>
<td><strong>$722.40</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

**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

```r
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.

```r
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.

```r
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 12.12  Creating a Customized Layout with BY Groups and ID Variables: Default

HTML5 Output

Program: Creating an HTML5 Report with the STYLE Option

```plaintext
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;
```

### Salary Expenses

<table>
<thead>
<tr>
<th>JobCode</th>
<th>Gender</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA1</td>
<td>F</td>
<td>$23,177.00</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>$22,454.00</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>$22,268.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>$67,899.00</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JobCode</th>
<th>Gender</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA2</td>
<td>F</td>
<td>$28,888.00</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>$27,787.00</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>$28,572.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>$85,247.00</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JobCode</th>
<th>Gender</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA3</td>
<td>F</td>
<td>$32,886.00</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>$33,419.00</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>$32,217.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>$98,522.00</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JobCode</th>
<th>Gender</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME1</td>
<td>M</td>
<td>$29,769.00</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td></td>
<td><strong>$281,437.00</strong></td>
</tr>
</tbody>
</table>
```
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;
run;

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 12.13 Creating a Customized Layout with BY Groups and ID Variables: HTML5 Output Using Styles

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

```sas
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 #
    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));
if final then call symput('num',put(n,8.));
```

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 #
  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 12.14**  Data Set PRINTLIB.DELAY

<table>
<thead>
<tr>
<th>flight</th>
<th>date</th>
<th>orig</th>
<th>dest</th>
<th>delaycat</th>
<th>destype</th>
<th>delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
<td>01MAR16</td>
<td>LGA</td>
<td>LAX</td>
<td>1-10 Minutes</td>
<td>Domestic</td>
<td>8</td>
</tr>
<tr>
<td>202</td>
<td>01MAR16</td>
<td>LGA</td>
<td>ORD</td>
<td>No Delay</td>
<td>Domestic</td>
<td>-5</td>
</tr>
<tr>
<td>219</td>
<td>01MAR16</td>
<td>LGA</td>
<td>LON</td>
<td>11+ Minutes</td>
<td>International</td>
<td>18</td>
</tr>
<tr>
<td>622</td>
<td>01MAR16</td>
<td>LGA</td>
<td>FRA</td>
<td>No Delay</td>
<td>International</td>
<td>-5</td>
</tr>
<tr>
<td>132</td>
<td>01MAR16</td>
<td>LGA</td>
<td>YYYZ</td>
<td>11+ Minutes</td>
<td>International</td>
<td>14</td>
</tr>
<tr>
<td>271</td>
<td>01MAR16</td>
<td>LGA</td>
<td>PAR</td>
<td>1-10 Minutes</td>
<td>International</td>
<td>5</td>
</tr>
<tr>
<td>302</td>
<td>01MAR16</td>
<td>LGA</td>
<td>WAS</td>
<td>No Delay</td>
<td>Domestic</td>
<td>-2</td>
</tr>
<tr>
<td>114</td>
<td>02MAR16</td>
<td>LGA</td>
<td>LAX</td>
<td>No Delay</td>
<td>Domestic</td>
<td>0</td>
</tr>
<tr>
<td>202</td>
<td>02MAR16</td>
<td>LGA</td>
<td>ORD</td>
<td>1-10 Minutes</td>
<td>Domestic</td>
<td>5</td>
</tr>
<tr>
<td>219</td>
<td>02MAR16</td>
<td>LGA</td>
<td>LON</td>
<td>11+ Minutes</td>
<td>International</td>
<td>18</td>
</tr>
</tbody>
</table>
### Output 12.15  Data Set PRINTLIB.INTERNAT

<table>
<thead>
<tr>
<th>flight</th>
<th>date</th>
<th>dest</th>
<th>boarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>219</td>
<td>01MAR16</td>
<td>LON</td>
<td>198</td>
</tr>
<tr>
<td>622</td>
<td>01MAR16</td>
<td>FRA</td>
<td>207</td>
</tr>
<tr>
<td>132</td>
<td>01MAR16</td>
<td>YYZ</td>
<td>115</td>
</tr>
<tr>
<td>271</td>
<td>01MAR16</td>
<td>PAR</td>
<td>138</td>
</tr>
<tr>
<td>219</td>
<td>02MAR16</td>
<td>LON</td>
<td>147</td>
</tr>
<tr>
<td>622</td>
<td>02MAR16</td>
<td>FRA</td>
<td>176</td>
</tr>
<tr>
<td>132</td>
<td>02MAR16</td>
<td>YYZ</td>
<td>106</td>
</tr>
<tr>
<td>271</td>
<td>02MAR16</td>
<td>PAR</td>
<td>172</td>
</tr>
<tr>
<td>219</td>
<td>03MAR16</td>
<td>LON</td>
<td>197</td>
</tr>
<tr>
<td>622</td>
<td>03MAR16</td>
<td>FRA</td>
<td>180</td>
</tr>
</tbody>
</table>
Chapter 13
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)>;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC PRINTTO</td>
<td>Define locations, other than ODS destinations, for SAS procedure output and for the SAS log</td>
<td>Ex. 1, Ex. 2</td>
</tr>
</tbody>
</table>
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 303

“Example 2: Routing to SAS Catalog Entries” on page 307

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 303

“Example 2: Routing to SAS Catalog Entries” on page 307
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 307

**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.
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.

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 302.

NEW clears any information that exists in a file and prepares the file to receive the SAS log or procedure output.

If you omit NEW, the new information is appended to the existing file.

If you specify both LOG= and PRINT=, NEW applies to both.

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.

tenry.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 302.

**UNIT=nn**

routes the output to the file identified by the fileref FTnnF001, 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 predefined 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 302.

---

### 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:

```sas
/* 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**

```sas
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
;```
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.

```sas
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.

```sas
proc printto log='log-file';
   run;
```

Create the NUMBERS data set. The DATA step uses list input to create the NUMBERS data set.

```sas
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.

```sas
proc printto print='output-file'
   new;
   run;
```

Print the NUMBERS data set. The PROC PRINT output is written to the specified external file.

```sas
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.

```sas
proc printto;
run;
```

Log

**Log 13.1  Portion of Log Routed to the Default Destination**

```sas
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 13.2  Portion of Log Routed to an External File

```sas
NOTE: PROCEDURE PRINTTO used (Total process time):
       real time           0.00 seconds
       cpu time            0.00 seconds

61     data numbers;
62        input x y z;
63     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     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     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 13.1  Procedure Output Routed to an External File

The Numbers Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14.2</td>
<td>25.2</td>
<td>96.8</td>
</tr>
<tr>
<td>2</td>
<td>10.8</td>
<td>51.6</td>
<td>96.8</td>
</tr>
<tr>
<td>3</td>
<td>9.5</td>
<td>34.2</td>
<td>138.2</td>
</tr>
<tr>
<td>4</td>
<td>8.8</td>
<td>27.6</td>
<td>83.2</td>
</tr>
<tr>
<td>5</td>
<td>11.5</td>
<td>49.4</td>
<td>287.0</td>
</tr>
<tr>
<td>6</td>
<td>6.3</td>
<td>42.0</td>
<td>170.7</td>
</tr>
</tbody>
</table>
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

```sas
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.

```sas
options source;
```

**Assign a libref.**

```sas
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 13.3  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         data lib1.inventry;
   63            length Dept $ 4 Item $ 6 Season $ 6 Year 4;
   64            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.INVENTORY 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         proc printto;
   82         run;
Output

Output 13.2  Procedure Output Routed to SAS Catalog Entry LIB1.CAT1.INVENTORY.OUTPUT.
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;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Example: Results from PROC PRODUCT_STATUS”</td>
<td>Specify the names and versions of the SAS Foundation products that are installed on your operating system.</td>
</tr>
</tbody>
</table>
### PRODUCT_STATUS Statement

Returns the names and versions of the SAS Foundation products that are installed on your operating system.

#### Syntax

```plaintext
PROC PRODUCT_STATUS;
```

#### Details

The PROC PRODUCT_STATUS statement does not have any arguments.

### Example: Results from PROC PRODUCT_STATUS

```plaintext
proc product_status;
run;
```

Here is a partial output that contains an example of the results that are produced by PROC PRODUCT_STATUS.

<table>
<thead>
<tr>
<th>Product Description</th>
<th>Custom Version Information</th>
<th>Image Version Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>For Base SAS Software</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For SAS/CONNECT</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For SAS/Secure 168-bit</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For SAS/ODBC Data Feeder</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For CAS Base Data Feeder</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For CAS Hive EmbProc DFeed</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For CAS Hive Data Feeder</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For CAS ODBC Data Feeder</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For CAS Oracle Data Feeder</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For CAS PCFile Data Feeder</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For CAS DFile Data Feeder</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For CAS DFile Data Feeder</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For TK Extensions for CAS DFeed</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For SAS/ACCESS Interface to PC Files</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For Cloud Analytic Services</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For Machine Learning Procedures</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For Common Analytic Procedures</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For Advanced Analytics Statistics</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>For Text Mining Process</td>
<td>V.03.00</td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom version information: 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 15
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>;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC PWENCODE</td>
<td>Encode a password</td>
<td>Ex. 1, Ex. 2, Ex. 3</td>
</tr>
</tbody>
</table>

PROC PWENCODE Statement

Encodes a password.

**Examples:**
- “Example 1: Encoding a Password” on page 316
- “Example 2: Saving an Encoded Password to the Paste Buffer” on page 316
- “Example 3: Specifying Method= SAS003 to Encode a Password” on page 317

**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 316
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>
<thead>
<tr>
<th>Encoding Method</th>
<th>Description</th>
<th>Supported Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>sas001</td>
<td>Uses base64 to encode passwords.</td>
<td>None</td>
</tr>
<tr>
<td>sas002, which can also be specified as sasenc</td>
<td>Uses a 32-bit key to encode passwords.</td>
<td>SASProprietary, which is supported in SAS software.</td>
</tr>
<tr>
<td>sas003</td>
<td>Uses a 256-bit key plus 16-bit salt to encode passwords.</td>
<td>AES (Advanced Encryption Standard), which is supported in SAS software.</td>
</tr>
<tr>
<td>sas004</td>
<td>Uses a 256-bit key plus 64-bit salt value to encode passwords.</td>
<td>AES (Advanced Encryption Standard), which is supported in SAS software.</td>
</tr>
</tbody>
</table>

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

```sas
proc pwencode in='my password';
run;
```

Program Description

Encode the password.

```sas
proc pwencode in='my password';
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log.

```sas
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.

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.

```sas
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=XXXXXXXXXXXX method=sas003;
29   run;

[SAS003]08D7B93810D390916F615117D71B2639B4BB
```

NOTE: PROCEDURE PWENCODE used (Total process time):
real time           0.00 seconds
cpu time            0.00 seconds
Chapter 16
SORT Procedure

Overview: SORT Procedure .................................................. 320
  What Does the SORT Procedure Do? .................................. 320
  Sorting SAS Data Sets ...................................................... 320

Concepts: SORT Procedure ................................................ 321
  Threaded Sorting .......................................................... 321
  Sorting Orders for Numeric Variables .................................. 322
  Sorting Orders for Character Variables ............................... 322
  Stored Sort Information .................................................. 324
  Presorted Input Data Sets ............................................... 324
  Linguistic Sorting of Data Sets and ICU ............................ 325
  Performance Tuning for PROC SORT ................................. 326
  Disk Space Considerations for PROC SORT ......................... 328

Syntax: SORT Procedure ...................................................... 329
  PROC SORT Statement ................................................... 330
  BY Statement ............................................................. 344
  KEY Statement ........................................................... 344

In-Database Processing: PROC SORT ..................................... 346

Integrity Constraints: SORT Procedure .................................. 348

Specifying the Host Sort Utility .......................................... 348
  Introduction to Using the Host Sort Utility ......................... 348
  Setting the Host Sort Utility as the Sort Algorithm .............. 348
  Sorting Based on Size or Observations .............................. 348
  Changing the Location of Temporary Files Used by the Host Sort Utility .............................. 349
  Passing Options to the Host Sort Utility ......................... 349
  Passing Parameters to the Host Sort Utility ..................... 349

Specifying the SORTSEQ= Option with a Host Sort Utility ....... 349

Results: SORT Procedure ..................................................... 350
  Procedure Output ....................................................... 350
  Output Data Set .......................................................... 350

Examples: SORT Procedure ................................................ 351
  Example 1: Sorting by the Values of Multiple Variables ....... 351
  Example 2: Sorting in Descending Order ............................ 353
  Example 3: Creating a View with a Single BY Variable .......... 355
  Example 4: Maintaining the Relative Order of Observations in Each BY Group ............................. 356
  Example 5: Retaining the First Observation of Each BY Group .... 359
  Example 6: Linguistic Sorting Using ALTERNATE_HANDLING= .... 361
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 350.

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:

```sas
proc sort data=employee;
   by idnumber;
run;

proc print data=employee;
run;
```

Log 16.1  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
```
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 353.

### Output 16.2 Observations Sorted by the Values of Three Variables

<table>
<thead>
<tr>
<th>Obs</th>
<th>Company</th>
<th>Town</th>
<th>Debt</th>
<th>Account Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paul's Pizza</td>
<td>Apex</td>
<td>83.00</td>
<td>1019</td>
</tr>
<tr>
<td>2</td>
<td>Peter's Auto Parts</td>
<td>Apex</td>
<td>65.79</td>
<td>7288</td>
</tr>
<tr>
<td>3</td>
<td>Watson Tabor Travel</td>
<td>Apex</td>
<td>37.95</td>
<td>3311</td>
</tr>
<tr>
<td>4</td>
<td>Tina's Pet Shop</td>
<td>Apex</td>
<td>37.95</td>
<td>5108</td>
</tr>
<tr>
<td>5</td>
<td>Apex Catering</td>
<td>Apex</td>
<td>37.95</td>
<td>9923</td>
</tr>
<tr>
<td>6</td>
<td>Deluxe Hardware</td>
<td>Garner</td>
<td>467.12</td>
<td>8941</td>
</tr>
<tr>
<td>7</td>
<td>Boyd &amp; Sons Accounting</td>
<td>Garner</td>
<td>312.49</td>
<td>4762</td>
</tr>
<tr>
<td>8</td>
<td>World Wide Electronics</td>
<td>Garner</td>
<td>119.95</td>
<td>1122</td>
</tr>
<tr>
<td>9</td>
<td>Elway Piano and Organ</td>
<td>Garner</td>
<td>65.79</td>
<td>5217</td>
</tr>
<tr>
<td>10</td>
<td>Ice Cream Delight</td>
<td>Holly Springs</td>
<td>299.98</td>
<td>2310</td>
</tr>
<tr>
<td>11</td>
<td>Tim's Burger Stand</td>
<td>Holly Springs</td>
<td>119.95</td>
<td>6335</td>
</tr>
<tr>
<td>12</td>
<td>Strickland Industries</td>
<td>Morrisville</td>
<td>657.22</td>
<td>1675</td>
</tr>
<tr>
<td>13</td>
<td>Pauline's Antiques</td>
<td>Morrisville</td>
<td>302.05</td>
<td>9112</td>
</tr>
<tr>
<td>14</td>
<td>Bob's Beds</td>
<td>Morrisville</td>
<td>119.95</td>
<td>4998</td>
</tr>
</tbody>
</table>

### 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 342 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 (NLS): 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>
<thead>
<tr>
<th>Table 16.1  ASCII Sort Order Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank ! &quot; # $ % &amp; ’ ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; &lt; = &gt; ? @</td>
</tr>
<tr>
<td>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 [ ] * _</td>
</tr>
<tr>
<td>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 { } ~</td>
</tr>
</tbody>
</table>
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:

```sas
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:

```sas
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 331.

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 (NLS): 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:

```sas
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:
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*.

To access the sort information that is stored with a data set, use the CONTENTS statement in PROC DATASETS.

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 339 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=sort-table | LINGUISTIC” in SAS Viya SQL Procedure User’s Guide.
• See PROC SORT option “LINGUISTIC<(collating-options)“ on page 333.
• See Chapter 3, “COPY Procedure,” on page 35.
• See the Appendix 2, “ICU License,” on page 439.

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 16, “SORT Procedure,” on page 320.

**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 DATA=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 346.

```
PROC SORT <collating-sequence-option> <other option(s)>;
    BY <DESCENDING> variable-1 <<DESCENDING> variable-2 ...>;
KEY variable(s) <\ option>;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC SORT</td>
<td>Order SAS data set observations by the values of one or more character or numeric variables</td>
<td>Ex. 1, Ex. 4, Ex. 5</td>
</tr>
<tr>
<td>BY</td>
<td>Specify the sorting variables</td>
<td>Ex. 1, Ex. 2, Ex. 5</td>
</tr>
<tr>
<td>KEY</td>
<td>Specify sorting keys and variables</td>
<td></td>
</tr>
</tbody>
</table>
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 351
- “Example 4: Maintaining the Relative Order of Observations in Each BY Group” on page 356
- “Example 5: Retaining the First Observation of Each BY Group” on page 359

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 322

DANISH
sorts characters according to the Danish and Norwegian convention.

The Danish and Norwegian collating sequence is shown in Figure 16.1 on page 332.

FINNISH
sorts characters according to the Finnish and Swedish convention.

The Finnish and Swedish collating sequence is shown in Figure 16.1 on page 332.
**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 16.1 on page 332.

**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 344 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 16.1 on page 332.

**SORTSEQ= collating-sequence**  
The collating-sequence can be one of the following:

- **collating-sequence-option** on page 332
- **translation_table** on page 332
- **encoding-value** on page 333
- **LINGUISTIC** on page 333


**Figure 16.1** National Collating Sequences of Alphanumeric Characters

Here are descriptions of the types of collating sequences:

- **collating-sequence-option**  
  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 331.

To see the Sorting Order of Character variables, “ASCII Order” on page 322 and Figure 16.1 on page 332 for all others.

**Example**
```sas
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 (NLS): 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 (NLS): 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 325.

**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_IGNOREABLE
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.


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 16.2 Arguments for CASE_FIRST=

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPER</td>
<td>Sorts uppercase letters first, then the lowercase letters.</td>
</tr>
<tr>
<td>LOWER</td>
<td>Sorts lowercase letters first, then the uppercase letters.</td>
</tr>
</tbody>
</table>

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 16.3 Values for COLLATION=

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRECT</td>
<td>Specifies a Hindi variant.</td>
</tr>
<tr>
<td>PHONEBOOK</td>
<td>Specifies a telephone-book style for ordering of characters. Select PHONEBOOK only with the German language.</td>
</tr>
<tr>
<td>PINYIN</td>
<td>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.</td>
</tr>
<tr>
<td>POSIX</td>
<td>Portable Operating System Interface. This option specifies a “C” locale ordering of characters.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STROKE</td>
<td>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.</td>
</tr>
<tr>
<td>TRADITIONAL</td>
<td>Specifies a traditional style for ordering of characters. For example, select TRADITIONAL with the Spanish Language.</td>
</tr>
</tbody>
</table>

**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 (NLS): 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 16.4 Values for NUMERIC_COLLATION**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Order numbers by the numeric value. For example, “8 Main St.” would sort before “45 Main St.”.</td>
</tr>
<tr>
<td>OFF</td>
<td>Order numbers by the character value. For example, &quot;45 Main St.&quot; would sort before &quot;8 Main St.&quot;.</td>
</tr>
</tbody>
</table>

**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 16.5 Values for STRENGTH=**

<table>
<thead>
<tr>
<th>Value</th>
<th>Type of Collation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMARY or 1</td>
<td>PRIMARY specifies differences between base characters (for example, &quot;a&quot; &lt; &quot;b&quot;).</td>
<td>It is the strongest difference. For example, dictionaries are divided into different sections by base character.</td>
</tr>
<tr>
<td>Value</td>
<td>Type of Collation</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SECONDARY</td>
<td>Accents in the characters are considered secondary differences (for example, &quot;as&quot; &lt; &quot;às&quot; &lt; &quot;at&quot;).</td>
<td>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.</td>
</tr>
<tr>
<td>or 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERTIARY</td>
<td>Upper and lowercase differences in characters are distinguished at the tertiary level (for example, &quot;ao&quot; &lt; &quot;Ao&quot; &lt; &quot;aò&quot;).</td>
<td>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.</td>
</tr>
<tr>
<td>or 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUATERNARY</td>
<td>When punctuation is ignored at level 1-3, an additional level can be used to distinguish words with and without punctuation (for example, &quot;a-b&quot; &lt; &quot;ab&quot; &lt; &quot;aB&quot;).</td>
<td>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.</td>
</tr>
<tr>
<td>or 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDENTICAL</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>or 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The SORTSEQ=LINGUISTIC option is available only on the PROC SORT SORTSEQ= option and is not available for the SAS System SORTSEQ= option.

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 325.

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.

For ICU License Agreement, see Appendix 2, “ICU License,” on page 439.

For more information, see “Specifying Linguistic Collation” in SAS Viya National Language Support (NLS): Reference Guide. For more information, see “Linguistic Sorting of Data Sets and ICU” on page 325.

**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.

In-database processing does not occur when the SORTSEQ= option is specified.

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 359

**NOEQUALS**

See “EQUALS | NOEQUALS” on page 338.

**NOTHREADS**

See “THREADS | NOTHREADS” on page 343.

**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 351

**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 339 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 338 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

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.

\( nK \)
specifies the amount of memory in kilobytes, where \( n \) is a real number.

\( nM \)
specifies the amount of memory in megabytes, where \( n \) is a real number.

\( nG \)
specifies the amount of memory in gigabytes, where \( n \) is a real number.

Specifying the \texttt{SORTSIZE=} option in the \texttt{PROC SORT} statement temporarily overrides the SAS system option. For more information, see “SORTSIZE= System Option” in \textit{SAS Viya System Options: Reference}.

\textbf{Alias} \hspace{1cm} \texttt{SIZE=}

\textbf{Default} \hspace{1cm} the value of the SAS system option \texttt{SORTSIZE=}

\textbf{Tips} Setting the \texttt{SORTSIZE=} option in the \texttt{PROC SORT} statement to \texttt{MAX} or \texttt{0}, or not setting the \texttt{SORTSIZE=} option, limits the \texttt{PROC SORT} to the available physical memory based on the settings of the SAS system options \texttt{REALMEMSIZE} and \texttt{MEMSIZE}.

For information about the SAS system options \texttt{REALMEMSIZE} and \texttt{MEMSIZE}, see the SAS documentation for your operating environment.

\textbf{TAGSORT}
stores only the BY variables and the observation numbers in temporary files. The \texttt{TAGSORT} option in the \texttt{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 \textit{tags}. At the completion of the sorting process, \texttt{PROC SORT} uses the tags to retrieve records from the input data set in sorted order.

\textit{Note}: The utility file created is much smaller than it would be if the \texttt{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 \texttt{WORK} system option. For more information, see “\texttt{WORK} System Option” in \textit{SAS Viya System Options: Reference}.

Note that while using the \texttt{TAGSORT} option might reduce temporary disk use, the processing time could be higher. However, on systems with limited available disk space, the \texttt{TAGSORT} option might enable data sets to be sorted in situations where that would otherwise not be possible.

\textbf{Restriction} \hspace{1cm} The \texttt{TAGSORT} option is not compatible with the \texttt{OVERWRITE} option.

\textbf{Interaction} \hspace{1cm} The \texttt{TAGSORT} option is not supported by the threaded sort.

\textbf{Tip} \hspace{1cm} When the total length of BY variables is small compared with the record length, \texttt{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 | NOTHREADES 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 321.

**UNIQUEOUT= SAS-data-set**

specifies the output data set for observations eliminated by the NOUNIQUEKEY option.

**Alias**

UNIOUT=
The DUPOUT= and UNIOUT= options are not compatible and cannot be specified simultaneously.

The UNIQUEOUT= option can be used with the NOUNIQUEKEY option. It cannot be combined with the NODUPKEY option.

The UNIQUEOUT= option can be used with the NOUNIQUEKEY option. It cannot be combined with the NODUPKEY option.

See “NOUNIQUEKEY” on page 340

SAS Viya Data Set Options: Reference

**BY Statement**

Specifies the sorting variables.

**Examples:**
- “Example 1: Sorting by the Values of Multiple Variables” on page 351
- “Example 2: Sorting in Descending Order” on page 353
- “Example 5: Retaining the First Observation of Each BY Group” on page 359

**Syntax**

```plaintext
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 353

**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:

```plaintext
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:

```plaintext
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:

```plaintext
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.

<table>
<thead>
<tr>
<th>Alias</th>
<th>ASC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>ASCENDING is the default sort order.</td>
</tr>
</tbody>
</table>
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
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 350.

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:

```sas
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:

```sas
options sortpgm=best sortcutp=40M;
```


### 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:

```sas
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 16, “SORT Procedure,” on page 320.

**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 355.

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>
<thead>
<tr>
<th>Task</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicit replacement of input data set</td>
<td>proc sort data=names;</td>
</tr>
<tr>
<td>explicit replacement of input data set</td>
<td>proc sort data=names out=names;</td>
</tr>
<tr>
<td>no replacement of input data set</td>
<td>proc sort data=names out=namesbyid;</td>
</tr>
</tbody>
</table>
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
```
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;
  by town company;
run;
```

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**.

```sas
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.

```sas
var company town debt accountnumber;
```

Specify the titles.

```sas
title 'Customers with Past-Due Accounts';
title2 'Listed Alphabetically within Town';
rn;
```

Output: HTML

**Output 16.3**  Sorting by the Values of Multiple Variables

<table>
<thead>
<tr>
<th>Obs</th>
<th>Company</th>
<th>Town</th>
<th>Debt</th>
<th>Accountnumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apex Catering</td>
<td>Apex</td>
<td>37.95</td>
<td>9923</td>
</tr>
<tr>
<td>2</td>
<td>Paul's Pizza</td>
<td>Apex</td>
<td>83.00</td>
<td>1019</td>
</tr>
<tr>
<td>3</td>
<td>Peter's Auto Parts</td>
<td>Apex</td>
<td>65.79</td>
<td>7238</td>
</tr>
<tr>
<td>4</td>
<td>Tina's Pet Shop</td>
<td>Apex</td>
<td>37.95</td>
<td>5168</td>
</tr>
<tr>
<td>5</td>
<td>Watson Tabor Travel</td>
<td>Apex</td>
<td>37.95</td>
<td>3131</td>
</tr>
<tr>
<td>6</td>
<td>Boyd &amp; Sons Accounting</td>
<td>Garner</td>
<td>312.49</td>
<td>4762</td>
</tr>
<tr>
<td>7</td>
<td>Deluxe Hardware</td>
<td>Garner</td>
<td>467.12</td>
<td>8941</td>
</tr>
<tr>
<td>8</td>
<td>Elway Piano and Organ</td>
<td>Garner</td>
<td>65.79</td>
<td>5217</td>
</tr>
<tr>
<td>9</td>
<td>World Wide Electronics</td>
<td>Garner</td>
<td>119.95</td>
<td>1122</td>
</tr>
<tr>
<td>10</td>
<td>Ice Cream Delight</td>
<td>Holly Springs</td>
<td>299.98</td>
<td>2310</td>
</tr>
<tr>
<td>11</td>
<td>Tim's Burger Stand</td>
<td>Holly Springs</td>
<td>119.95</td>
<td>6335</td>
</tr>
<tr>
<td>12</td>
<td>Bob's Beds</td>
<td>Morrisville</td>
<td>119.95</td>
<td>4988</td>
</tr>
<tr>
<td>13</td>
<td>Pauline's Antiques</td>
<td>Morrisville</td>
<td>302.05</td>
<td>9112</td>
</tr>
<tr>
<td>14</td>
<td>Strickland Industries</td>
<td>Morrisville</td>
<td>657.22</td>
<td>1675</td>
</tr>
</tbody>
</table>

Example 2: Sorting in Descending Order

**Features:**  This example BY statement option DESCENDING

---

### Example 2: Sorting in Descending Order

**Features:**  This example BY statement option DESCENDING
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 16.4  Sorting in Descending Order

<table>
<thead>
<tr>
<th>Obs</th>
<th>Company</th>
<th>Town</th>
<th>Debt</th>
<th>AccountNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paul’s Pizza</td>
<td>Apex</td>
<td>83.00</td>
<td>1019</td>
</tr>
<tr>
<td>2</td>
<td>Peter’s Auto Parts</td>
<td>Apex</td>
<td>65.79</td>
<td>7238</td>
</tr>
<tr>
<td>3</td>
<td>Watson Tabor Travel</td>
<td>Apex</td>
<td>37.95</td>
<td>3131</td>
</tr>
<tr>
<td>4</td>
<td>Tina’s Pet Shop</td>
<td>Apex</td>
<td>37.95</td>
<td>5108</td>
</tr>
<tr>
<td>5</td>
<td>Apex Catering</td>
<td>Apex</td>
<td>37.95</td>
<td>9923</td>
</tr>
<tr>
<td>6</td>
<td>Deluxe Hardware</td>
<td>Garner</td>
<td>467.12</td>
<td>8941</td>
</tr>
<tr>
<td>7</td>
<td>Boyd &amp; Sons Accounting</td>
<td>Garner</td>
<td>312.49</td>
<td>4792</td>
</tr>
<tr>
<td>8</td>
<td>World Wide Electronics</td>
<td>Garner</td>
<td>119.95</td>
<td>1122</td>
</tr>
<tr>
<td>9</td>
<td>Elway Piano and Organ</td>
<td>Garner</td>
<td>65.79</td>
<td>5217</td>
</tr>
<tr>
<td>10</td>
<td>Ice Cream Delight</td>
<td>Holly Springs</td>
<td>299.96</td>
<td>2310</td>
</tr>
<tr>
<td>11</td>
<td>Tim’s Burger Stand</td>
<td>Holly Springs</td>
<td>119.95</td>
<td>6335</td>
</tr>
<tr>
<td>12</td>
<td>Strickland Industries</td>
<td>Morrisville</td>
<td>657.22</td>
<td>1675</td>
</tr>
<tr>
<td>13</td>
<td>Pauline’s Antiques</td>
<td>Morrisville</td>
<td>302.05</td>
<td>9112</td>
</tr>
<tr>
<td>14</td>
<td>Bob’s Beds</td>
<td>Morrisville</td>
<td>119.95</td>
<td>4998</td>
</tr>
</tbody>
</table>

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
```
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.
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

```sas
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.

Sort by one variable. The BY statement specifies that observations should be ordered by town.

```sas
by town;
run;
```

Print the output data set TOWNS. PROC PRINT prints the data set TOWNS.

```sas
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.

```sas
   var town company debt accountnumber;
```

Specify the title.

```sas
   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 16.8  Retaining the First Observation of Each BY Group

<table>
<thead>
<tr>
<th>Town</th>
<th>Company</th>
<th>Debt</th>
<th>AccountNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apex</td>
<td>Paul's Pizza</td>
<td>83.00</td>
<td>1019</td>
</tr>
<tr>
<td>Garner</td>
<td>World Wide Electronics</td>
<td>119.95</td>
<td>1122</td>
</tr>
<tr>
<td>Holly Springs</td>
<td>Ice Cream Delight</td>
<td>299.98</td>
<td>2310</td>
</tr>
<tr>
<td>Morrisville</td>
<td>Strickland Industries</td>
<td>657.22</td>
<td>1675</td>
</tr>
</tbody>
</table>

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 363.

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 363 shows how to distinguish the strings further.

Program
```
data a;
  length x $ 10;
```
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;
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 363 shows how to distinguish the strings more.

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**

```sas
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;
```

**Program Description**

__Create the data set.__

```sas
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.

```sas
proc sort data=a sortseq=linguistic( ALTERNATE_HANDLING=SHIFTED STRENGTH=4);
  by x;
run;
```

__Print the output data set A.__ The TITLE statement tells the PRINT procedure the title to use for the output. PROC PRINT then prints data set A.

```sas
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.

```sas
Title1 "Linguistic Collation with STRENGTH=4 and BY Processing";
proc print data=a;
  var x;
  by x;
run;
```
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 16.10 Linguistic Sorting Using the ALTERNATE_HANDLING and STRENGTH Options**

Linguistic Collation with STRENGTH=4

<table>
<thead>
<tr>
<th>Obs</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a-b</td>
</tr>
<tr>
<td>2</td>
<td>a-b</td>
</tr>
<tr>
<td>3</td>
<td>ab</td>
</tr>
<tr>
<td>4</td>
<td>aB</td>
</tr>
</tbody>
</table>

Linguistic Collation with STRENGTH=4 and BY Processing

- **x=a-b**
  
<table>
<thead>
<tr>
<th>Obs</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a-b</td>
</tr>
<tr>
<td>2</td>
<td>a-b</td>
</tr>
</tbody>
</table>

- **x=ab**
  
<table>
<thead>
<tr>
<th>Obs</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ab</td>
</tr>
</tbody>
</table>

- **x=aB**
  
<table>
<thead>
<tr>
<th>Obs</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>aB</td>
</tr>
</tbody>
</table>
Chapter 17
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 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.

```sas
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 17.1 A Simple Transposition**

<table>
<thead>
<tr>
<th>The Input Data Set</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester1 Tester2 Tester3 Tester4</td>
<td></td>
</tr>
<tr>
<td>22  25  21  21</td>
<td></td>
</tr>
<tr>
<td>15  19  18  17</td>
<td></td>
</tr>
<tr>
<td>17  19  19  19</td>
<td></td>
</tr>
<tr>
<td>20  19  16  19</td>
<td></td>
</tr>
<tr>
<td>14  15  13  13</td>
<td></td>
</tr>
<tr>
<td>15  17  18  19</td>
<td></td>
</tr>
<tr>
<td>10  11  9  10</td>
<td></td>
</tr>
<tr>
<td>22  24  23  21</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The Output Data Set</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em> COL1 COL2 COL3 COL4 COL5 COL6 COL7 COL8</td>
<td></td>
</tr>
<tr>
<td>Tester1 22 15 17 20 14 15 10 22</td>
<td></td>
</tr>
<tr>
<td>Tester2 25 19 19 19 15 17 11 24</td>
<td></td>
</tr>
<tr>
<td>Tester3 21 18 19 16 13 18 9 23</td>
<td></td>
</tr>
<tr>
<td>Tester4 21 17 19 19 13 19 10 21</td>
<td></td>
</tr>
</tbody>
</table>
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 17.2 A Transposition with BY Groups

<table>
<thead>
<tr>
<th>Input Data Set</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L W L W L W L W</td>
</tr>
<tr>
<td>o</td>
<td>e e e e e e e e</td>
</tr>
<tr>
<td>c</td>
<td>n i n i n i n i</td>
</tr>
<tr>
<td>a</td>
<td>t h g g g g g g</td>
</tr>
<tr>
<td>t</td>
<td>a t h t h t h t h</td>
</tr>
<tr>
<td>o</td>
<td>t h t h t h t h</td>
</tr>
<tr>
<td>n</td>
<td>e 1 1 2 2 3 3 4 4</td>
</tr>
<tr>
<td>Cole Pond 02JUN95</td>
<td>31 0.25 32 0.30 32 0.25 33 0.30</td>
</tr>
<tr>
<td>Cole Pond 03JUL95</td>
<td>33 0.32 34 0.41 37 0.48 32 0.28</td>
</tr>
<tr>
<td>Cole Pond 04AUG95</td>
<td>29 0.23 30 0.25 34 0.47 32 0.30</td>
</tr>
<tr>
<td>Eagle Lake 02JUN95</td>
<td>32 0.35 32 0.25 33 0.30</td>
</tr>
<tr>
<td>Eagle Lake 03JUL95</td>
<td>30 0.20 36 0.45</td>
</tr>
<tr>
<td>Eagle Lake 04AUG95</td>
<td>33 0.30 33 0.28 34 0.42</td>
</tr>
</tbody>
</table>

Fish Length Data for Each Location and Date | 2 |

<table>
<thead>
<tr>
<th>Location</th>
<th>Date</th>
<th><em>NAME</em></th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cole Pond</td>
<td>02JUN95</td>
<td>Length1</td>
<td>31</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>02JUN95</td>
<td>Length2</td>
<td>32</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>02JUN95</td>
<td>Length3</td>
<td>32</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>02JUN95</td>
<td>Length4</td>
<td>33</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>03JUL95</td>
<td>Length1</td>
<td>33</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>03JUL95</td>
<td>Length2</td>
<td>34</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>03JUL95</td>
<td>Length3</td>
<td>37</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>03JUL95</td>
<td>Length4</td>
<td>32</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>04AUG95</td>
<td>Length1</td>
<td>29</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>04AUG95</td>
<td>Length2</td>
<td>30</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>04AUG95</td>
<td>Length3</td>
<td>34</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>04AUG95</td>
<td>Length4</td>
<td>32</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>02JUN95</td>
<td>Length1</td>
<td>32</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>02JUN95</td>
<td>Length2</td>
<td>32</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>02JUN95</td>
<td>Length3</td>
<td>33</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>02JUN95</td>
<td>Length4</td>
<td>.</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>03JUL95</td>
<td>Length1</td>
<td>30</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>03JUL95</td>
<td>Length2</td>
<td>36</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>03JUL95</td>
<td>Length3</td>
<td>.</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>03JUL95</td>
<td>Length4</td>
<td>.</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>04AUG95</td>
<td>Length1</td>
<td>33</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>04AUG95</td>
<td>Length2</td>
<td>33</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>04AUG95</td>
<td>Length3</td>
<td>34</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>04AUG95</td>
<td>Length4</td>
<td>.</td>
</tr>
</tbody>
</table>
In-Database Processing for PROC TRANSPOSE (Preproduction)

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 data source. Faster processing is possible because data is manipulated locally, on the data source, using high-speed secondary storage devices instead of being transported across a relatively slow network connection. The data source is used because it might have more processing resources at its disposal, and it might be capable of optimizing a query for execution in a highly parallel and scalable fashion.

When the DATA= input table is stored as a table or view in a database, PROC TRANSPOSE can use in-database processing to perform most of its work within the database. 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 TRANSPOSE supports the following data providers:

- Hadoop
- Teradata

The TRANSPOSE procedure performs a dynamic transformation of the data in which the characteristics of the output table, specifically the number and names of the variables, as well as their types, are determined from the variable values, as well as the characteristics of the input table. This dynamic behavior is achieved by a two-pass process in which rows of the input table are examined to determine the characteristics of the output table, during the first pass, and in which the work of transposing the data is performed during the second pass. Parallel operation within the massively parallel processing (MPP) database speeds up both the first and second pass. The MPP performs a set of coordinated computations in parallel with the use of a large number of processors or separate computers. In the first pass, rows are examined in parallel and in place, according to the manner in which they are already partitioned across the nodes of a cluster. In the second pass, rows are repartitioned to form BY groups that are then processed independently and in parallel.

Both the first and second passes of the in-database processing are performed by executing a DS2 program within the SAS Embedded Process that resides within the nodes of the cluster. The DBMS provides the SAS Embedded Process with the ability to read data from and write data to tables. The SAS Embedded Process provides an execution context for the DS2 program. Because the two passes of work are expressed in the DS2 language, columns of the tables are cast to variables that have DS2 data types. Data type support within DS2 is more extensive than that provided by the traditional SAS system so, when executing inside the database, the ability of the TRANSPOSE procedure to preserve data types and value of input data within the transposed output data is enhanced.

The SQLGENERATION system option or LIBNAME statement option controls whether and how in-database procedures are run inside the database. PROC TRANSPOSE runs inside the database when you specify INDB=YES. There are many programming considerations that can prevent in-database processing. For a complete listing, see “Procedure Considerations and Limitations” in SAS In-Database Products: User's Guide.
**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. An example that is not supported is

```sas
PROC TRANSPOSE DATA=CAS.FOO OUT=CAS.FOO
```

You can point to CAS with one of the options. For example:

```sas
PROC TRANSPOSE DATA=CAS.FOO OUT=WORK.BAR
```

**Tips:** Does not support the Output Delivery System

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

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**

```sas
PROC TRANSPOSE <DATA=input-data-set> <DELIMITER=delimiter> <LABEL=label> <LET> <NAME=name> <OUT=output-data-set> <PREFIX=prefix> <SUFFIX=suffix>;
```

**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 380
“Example 2: Naming Transposed Variables” on page 381
“Example 3: Labeling Transposed Variables” on page 383
“Example 4: Transposing BY Groups” on page 385
“Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values” on page 387
“Example 6: Transposing Data for Statistical Analysis” on page 389

Syntax

PROC TRANSPOSE <DATA= input-data-set> <DELIMITER= delimiter <INDB= YES | NO>>
> <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 376

INDB= YES| NO
specifies if in-database processing is active. YES specifies that INDB is active. YES is the default. NO specifies that INDB is not active.

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 387

**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 381

**OUT= output-data-set**

names the output data set. If output-data-set does not exist, then PROC TRANSPOSE creates it by using the DATAn naming convention.

Default DATAn

See “Example 1: Performing a Simple Transposition” on page 380

**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, ..., VARn.

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 381

**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 385

**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**

PROC TRANSPOSE does not transpose BY groups. Instead, for each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes.

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 17.1  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.

**Example:**  “Example 6: Transposing Data for Statistical Analysis” on page 389

**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 381

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 a warning message about duplicate ID values and stop processing. However, if the LET option is specified in the PROC TRANSPOSE statement then the procedure issues a warning message and continue 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 combined with the PREFIX, DELIMITER, and SUFFIX option values can cause duplicate output data set variable names in cases where 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 383

---

### 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 385

“Example 6: Transposing Data for Statistical Analysis” on page 389

---

**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 DELIMITER= 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, …, COLn 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**

```sas
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.

```sas
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.

```sas
proc transpose data=score out=score_transposed;
run;
```

Print the SCORE_TRANSPOSED data set. The NOOBS option suppresses the printing of observation numbers.

```sas
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 17.3 Student Test Scores in Variables

```
  _NAME_  COL1  COL2  COL3  COL4  COL5  COL6  COL7
  Test1   94    51    95    61    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=
This example uses the values of a variable and a user-supplied value to name transposed variables.

Program

```sas
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.

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.

Print the IDNUMBER data set. The NOOBS option suppresses the printing of observation numbers.

Output

The following data set is the output data set, IDNUMBER.
Output 17.4  Student Test Scores

<table>
<thead>
<tr>
<th>Student Test Scores</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>sn0545</td>
</tr>
<tr>
<td>Test1</td>
<td>94</td>
</tr>
<tr>
<td>Test2</td>
<td>91</td>
</tr>
<tr>
<td>Final</td>
<td>87</td>
</tr>
</tbody>
</table>

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**

```sas
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.

```sas
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.

```sas
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

```sas
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.

```sas
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.

```sas
proc contents data=idlabel;
run;
```

Output 1

This data set is the output data set, IDLABEL.

**Output 17.5 Student Test Scores, IDLABEL**

<table>
<thead>
<tr>
<th></th>
<th>Capalleti</th>
<th>Dubose</th>
<th>Engles</th>
<th>Grant</th>
<th>Krupski</th>
<th>Lundsford</th>
<th>McBane</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>94</td>
<td>51</td>
<td>95</td>
<td>63</td>
<td>80</td>
<td>92</td>
<td>75</td>
</tr>
<tr>
<td>Test2</td>
<td>91</td>
<td>65</td>
<td>97</td>
<td>75</td>
<td>76</td>
<td>40</td>
<td>78</td>
</tr>
<tr>
<td>Final</td>
<td>87</td>
<td>91</td>
<td>97</td>
<td>80</td>
<td>71</td>
<td>86</td>
<td>72</td>
</tr>
</tbody>
</table>

Program 2

```sas
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.

```sas
proc contents data=idlabel;
run;
```
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 FISHLNGTH 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 17.7  Fish Length Data

<table>
<thead>
<tr>
<th>Location</th>
<th>Date</th>
<th><em>NAME</em></th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cole Pond</td>
<td>02JUN95</td>
<td>Length1</td>
<td>31</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>02JUN95</td>
<td>Length2</td>
<td>32</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>02JUN95</td>
<td>Length3</td>
<td>32</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>02JUN95</td>
<td>Length4</td>
<td>33</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>03JUL95</td>
<td>Length1</td>
<td>33</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>03JUL95</td>
<td>Length2</td>
<td>34</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>03JUL95</td>
<td>Length3</td>
<td>37</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>03JUL95</td>
<td>Length4</td>
<td>32</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>04AUG95</td>
<td>Length1</td>
<td>29</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>04AUG95</td>
<td>Length2</td>
<td>30</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>04AUG95</td>
<td>Length3</td>
<td>34</td>
</tr>
<tr>
<td>Cole Pond</td>
<td>04AUG95</td>
<td>Length4</td>
<td>32</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>02JUN95</td>
<td>Length1</td>
<td>32</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>02JUN95</td>
<td>Length2</td>
<td>32</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>02JUN95</td>
<td>Length3</td>
<td>33</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>02JUN95</td>
<td>Length4</td>
<td>.</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>03JUL95</td>
<td>Length1</td>
<td>30</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>03JUL95</td>
<td>Length2</td>
<td>36</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>03JUL95</td>
<td>Length3</td>
<td>.</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>03JUL95</td>
<td>Length4</td>
<td>.</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>04AUG95</td>
<td>Length1</td>
<td>33</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>04AUG95</td>
<td>Length2</td>
<td>33</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>04AUG95</td>
<td>Length3</td>
<td>34</td>
</tr>
<tr>
<td>Eagle Lake</td>
<td>04AUG95</td>
<td>Length4</td>
<td>.</td>
</tr>
</tbody>
</table>

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.

```plaintext
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.

```plaintext
by company;
```

**Name the transposed variables.** The values of Date are used as names for the transposed variables.

```plaintext
id date;
run;
```

**Print the CLOSE data set.** The NOOBS option suppresses the printing of observation numbers.

```plaintext
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 17.8  Closing Prices**

```
Closing Prices for Horizon Kites and SkyHi Kites

<table>
<thead>
<tr>
<th>Company</th>
<th>Price</th>
<th>jun11</th>
<th>jun12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizon Kites</td>
<td>Price</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>SkyHi Kites</td>
<td>Price</td>
<td>44</td>
<td>45</td>
</tr>
</tbody>
</table>
```

---

**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.


**Program 1**

```plaintext
options nodate pageno=1 linesize=80 pagesize=40;
data weights;
   input Program $ s1-s7;
```
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 76 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;
```
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.

```sas
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.

```sas
proc print data=split(obs=15) noobs;
  title 'SPLIT Data Set';
  title2 'First 15 Observations Only';
run;
```
Output 1

Output 17.9  Split Data Set

<table>
<thead>
<tr>
<th>Program</th>
<th>Subject</th>
<th>Time</th>
<th>Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONT</td>
<td>1</td>
<td>1</td>
<td>85</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>2</td>
<td>85</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>3</td>
<td>86</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>4</td>
<td>85</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>5</td>
<td>87</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>6</td>
<td>86</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>7</td>
<td>87</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>2</td>
<td>79</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>3</td>
<td>79</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>4</td>
<td>78</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>5</td>
<td>78</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>6</td>
<td>79</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>7</td>
<td>78</td>
</tr>
<tr>
<td>CONT</td>
<td>3</td>
<td>1</td>
<td>78</td>
</tr>
</tbody>
</table>

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 TOTSPILIT. The TOTSPILIT data set contains the same variables as SPLIT and a variable for each strength measurement (Str1-Str7). TOTSPILIT 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. TOTSPILIT 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 17.10  TOTSPLIT Data Set**

<table>
<thead>
<tr>
<th>Program</th>
<th>Subject</th>
<th>Time</th>
<th>Strength</th>
<th><em>NAME</em></th>
<th>Str1</th>
<th>Str2</th>
<th>Str3</th>
<th>Str4</th>
<th>Str5</th>
<th>Str6</th>
<th>Str7</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONT</td>
<td>1</td>
<td>1</td>
<td>85</td>
<td>_</td>
<td>85</td>
<td>85</td>
<td>86</td>
<td>85</td>
<td>87</td>
<td>86</td>
<td>87</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>2</td>
<td>85</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>3</td>
<td>86</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>4</td>
<td>85</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>5</td>
<td>87</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>6</td>
<td>86</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>1</td>
<td>7</td>
<td>87</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>1</td>
<td>80</td>
<td>_</td>
<td>80</td>
<td>79</td>
<td>79</td>
<td>78</td>
<td>78</td>
<td>79</td>
<td>78</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>2</td>
<td>79</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>3</td>
<td>79</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>4</td>
<td>78</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>5</td>
<td>78</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>6</td>
<td>79</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>2</td>
<td>7</td>
<td>78</td>
<td></td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>CONT</td>
<td>3</td>
<td>1</td>
<td>78</td>
<td>_</td>
<td>78</td>
<td>77</td>
<td>77</td>
<td>77</td>
<td>76</td>
<td>76</td>
<td>77</td>
</tr>
</tbody>
</table>
Appendixes

Appendix 1
Raw Data and DATA Steps for Base SAS Procedures ............. 397

Appendix 2
ICU License ............................................................... 439
Appendix 1
Raw Data and DATA Steps for Base SAS Procedures

Overview of Raw Data and DATA Steps for Base SAS Procedures ........... 398
AddressPrint ............................................................... 398
CARSURVEY ............................................................ 399
CENSUS ................................................................. 400
CHARITY ................................................................. 401
CUSTOMER_RESPONSE ............................................. 403
DJIA ................................................................. 406
EDUCATION .......................................................... 407
EMPDATA .............................................................. 407
ENERGY ............................................................... 410
EXP Library ......................................................... 412
   EXP.RESULTS .................................................. 412
   EXP.SUR .......................................................... 412
EXPREV .............................................................. 413
GROC ................................................................. 414
MATCH_11 ............................................................. 415
PROCCLIB.DELAY .................................................. 416
PROCCLIB.EMP95 ................................................ 417
PROCCLIB.EMP96 ................................................ 418
PROCCLIB.INTERNAT .......................................... 419
PROCCLIB.LAKES .................................................. 419
PROCCLIB.PAYROLL .............................................. 420
PROCCLIB.STAFF .................................................. 423
RADIO ................................................................. 426
SALES ................................................................. 438
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 W Jackson Blvd:San Jose:CA:408-752-3500:leota@hotmail.com:
Josephine:Darakjy:4 B Blue Ridge Blvd:Brighton:MI:810-292-9388:josephine_darakjy@darakjy.org:
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 W Jackson Blvd:San Jose:CA:408-752-3500:leota@hotmail.com:

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  83  75  82
 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
CENSUS

data census;
   input Density CrimeRate State $ 14-27 PostalCode $ 29-30;
datalines;
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
<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Number</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Randall</td>
<td>12.98</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Sam</td>
<td>15.88</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Tyra</td>
<td>21.88</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Myrtle</td>
<td>47.33</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Frank</td>
<td>41.11</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Cameron</td>
<td>65.44</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Vern</td>
<td>17.89</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Wendell</td>
<td>23.00</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Bob</td>
<td>26.88</td>
</tr>
<tr>
<td>Monroe</td>
<td>2008</td>
<td>Leah</td>
<td>28.99</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Becky</td>
<td>30.33</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Myrtle</td>
<td>35.75</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Edgar</td>
<td>27.11</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Dawson</td>
<td>17.24</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Lou</td>
<td>5.12</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Damien</td>
<td>18.74</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Mona</td>
<td>27.43</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Della</td>
<td>56.78</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Monique</td>
<td>29.88</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Carl</td>
<td>31.12</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Reba</td>
<td>35.16</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Dax</td>
<td>27.65</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Gary</td>
<td>23.11</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Suzie</td>
<td>26.65</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Benito</td>
<td>27.98</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Thomas</td>
<td>21.99</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Annie</td>
<td>24.99</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Paul</td>
<td>27.98</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Alex</td>
<td>24.00</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Lauren</td>
<td>15.00</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Julia</td>
<td>12.98</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Keith</td>
<td>11.89</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Jackie</td>
<td>26.88</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Pablo</td>
<td>13.98</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>L.T.</td>
<td>56.87</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Willard</td>
<td>78.65</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Kathy</td>
<td>32.88</td>
</tr>
<tr>
<td>Monroe</td>
<td>2009</td>
<td>Abby</td>
<td>35.88</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Arturo</td>
<td>34.98</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Grace</td>
<td>27.55</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Winston</td>
<td>23.88</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Vince</td>
<td>12.88</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Claude</td>
<td>15.62</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Mary</td>
<td>28.99</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Abner</td>
<td>25.89</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Jay</td>
<td>35.89</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Alicia</td>
<td>28.77</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Freddy</td>
<td>29.00</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Eloise</td>
<td>31.67</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Jenny</td>
<td>43.89</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Thelma</td>
<td>52.63</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Tina</td>
<td>19.67</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Eric</td>
<td>24.89</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Bubba</td>
<td>37.88</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>G.L.</td>
<td>25.89</td>
</tr>
<tr>
<td>Kennedy</td>
<td>2007</td>
<td>Bert</td>
<td>28.89</td>
</tr>
</tbody>
</table>
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

DATA CUSTOMER_RESPONSE

data customer_response;
input Customer Factor1-Factor4 Source1-Source3 Quality1-Quality3;
datalines;
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 . . . .
8 1 . 1 1 . 1 1 1.
9 1 1 . 1 1 . . . 1
1 0 1 . . 1 1 . 1 1.

CUSTOMER_RESPONSE
data djia;
  input Year HighDate date7. High LowDate date7. Low;
  format highdate lowdate date7.;
datalines;
1968 03DEC68  985.21  21MAR68  825.13
1969 04MAY69  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  3812.08
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  20OCT02  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
;
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     NJ 20.4 7549 269 NB
  New Mexico     NM 28.5 3473 256 W
  New York       NY 35.0 .   261 NB
  North Carolina NC 31.2 3874 250 SE
  North Dakota   ND 12.1 3952 281 MW
  Ohio           OH 24.4 4649 264 MW
  ;

data empdata;
  input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
<table>
<thead>
<tr>
<th>ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>City</th>
<th>State</th>
<th>Gender</th>
<th>JobCode</th>
<th>HomePhone</th>
<th>Birth Date</th>
<th>Hired Date</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adams</td>
<td>Gerald</td>
<td>Stamford</td>
<td>CT</td>
<td>M</td>
<td>TA2</td>
<td>34376</td>
<td>15SEP70</td>
<td>07JUN05</td>
<td>203-781-1255</td>
</tr>
<tr>
<td>1653</td>
<td>Alexander</td>
<td>Susan</td>
<td>Bridgeport</td>
<td>CT</td>
<td>F</td>
<td>ME2</td>
<td>35108</td>
<td>18OCT72</td>
<td>12AUG98</td>
<td>203-675-7715</td>
</tr>
<tr>
<td>1400</td>
<td>Apple</td>
<td>Troy</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>ME1</td>
<td>29769</td>
<td>08NOV85</td>
<td>19OCT06</td>
<td>212-586-0808</td>
</tr>
<tr>
<td>1350</td>
<td>Arthur</td>
<td>Barbara</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>PA3</td>
<td>32886</td>
<td>03SEP63</td>
<td>01AUG00</td>
<td>718-383-1549</td>
</tr>
<tr>
<td>1401</td>
<td>Avery</td>
<td>Jerry</td>
<td>Paterson</td>
<td>NJ</td>
<td>M</td>
<td>TA3</td>
<td>38822</td>
<td>16DEC68</td>
<td>20NOV93</td>
<td>201-732-8787</td>
</tr>
<tr>
<td>1499</td>
<td>Barefoot</td>
<td>Joseph</td>
<td>Princeton</td>
<td>NJ</td>
<td>M</td>
<td>ME3</td>
<td>43025</td>
<td>29APR62</td>
<td>10JUN95</td>
<td>201-812-5665</td>
</tr>
<tr>
<td>1101</td>
<td>Baucom</td>
<td>Walter</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>SCP</td>
<td>18723</td>
<td>09JUN80</td>
<td>04OCT98</td>
<td>212-586-8060</td>
</tr>
<tr>
<td>1333</td>
<td>Blair</td>
<td>Justin</td>
<td>Stamford</td>
<td>CT</td>
<td>M</td>
<td>PT2</td>
<td>88606</td>
<td>02APR79</td>
<td>13FEB03</td>
<td>203-781-1777</td>
</tr>
<tr>
<td>1402</td>
<td>Blalock</td>
<td>Ralph</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>TA2</td>
<td>32615</td>
<td>20JAN71</td>
<td>05DEC98</td>
<td>718-384-2849</td>
</tr>
<tr>
<td>1479</td>
<td>Bostic</td>
<td>Marie</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>TA3</td>
<td>38785</td>
<td>25DEC66</td>
<td>08OCT03</td>
<td>718-384-8816</td>
</tr>
<tr>
<td>1403</td>
<td>Bowden</td>
<td>Earl</td>
<td>Bridgeport</td>
<td>CT</td>
<td>M</td>
<td>ME1</td>
<td>28072</td>
<td>31JAN79</td>
<td>24DEC99</td>
<td>203-675-3434</td>
</tr>
<tr>
<td>1739</td>
<td>Boyce</td>
<td>Jonathan</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>PT1</td>
<td>66517</td>
<td>28DEC82</td>
<td>30JAN00</td>
<td>212-587-1247</td>
</tr>
<tr>
<td>1658</td>
<td>Bradley</td>
<td>Jeremy</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>SCP</td>
<td>17943</td>
<td>11APR65</td>
<td>03MAR00</td>
<td>212-587-3622</td>
</tr>
<tr>
<td>1428</td>
<td>Brady</td>
<td>Christine</td>
<td>Stamford</td>
<td>CT</td>
<td>F</td>
<td>TA3</td>
<td>68767</td>
<td>07APR80</td>
<td>19NOV02</td>
<td>203-781-1212</td>
</tr>
<tr>
<td>1782</td>
<td>Brown</td>
<td>Jason</td>
<td>Stamford</td>
<td>CT</td>
<td>M</td>
<td>ME2</td>
<td>35345</td>
<td>07DEC73</td>
<td>25FEB00</td>
<td>203-781-0019</td>
</tr>
<tr>
<td>1244</td>
<td>Bryant</td>
<td>Leonard</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>ME2</td>
<td>36925</td>
<td>03SEP71</td>
<td>20JAN96</td>
<td>718-383-3334</td>
</tr>
<tr>
<td>1383</td>
<td>Burnette</td>
<td>Thomas</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>BCK</td>
<td>25823</td>
<td>28JAN76</td>
<td>23OCT00</td>
<td>718-384-3569</td>
</tr>
<tr>
<td>1574</td>
<td>Cahill</td>
<td>Marshall</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>PA2</td>
<td>28572</td>
<td>30APR74</td>
<td>23DEC97</td>
<td>718-383-2338</td>
</tr>
<tr>
<td>1789</td>
<td>Caraway</td>
<td>Davis</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>SCP</td>
<td>18326</td>
<td>28JAN85</td>
<td>14APR04</td>
<td>212-587-9000</td>
</tr>
<tr>
<td>1404</td>
<td>Carter</td>
<td>Donald</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>PT2</td>
<td>91376</td>
<td>27FEB71</td>
<td>04JAN98</td>
<td>718-384-2946</td>
</tr>
<tr>
<td>1437</td>
<td>Carter</td>
<td>Dorothy</td>
<td>Bridgeport</td>
<td>CT</td>
<td>F</td>
<td>A3</td>
<td>33104</td>
<td>23SEP68</td>
<td>03SEP92</td>
<td>203-675-4117</td>
</tr>
<tr>
<td>1639</td>
<td>Carter</td>
<td>Karen</td>
<td>Stamford</td>
<td>CT</td>
<td>M</td>
<td>A3</td>
<td>40260</td>
<td>29JUN65</td>
<td>31JAN92</td>
<td>203-781-8839</td>
</tr>
<tr>
<td>1269</td>
<td>Caston</td>
<td>Franklin</td>
<td>Stamford</td>
<td>CT</td>
<td>M</td>
<td>NA1</td>
<td>41690</td>
<td>06MAY80</td>
<td>01DEC00</td>
<td>203-781-3335</td>
</tr>
<tr>
<td>1065</td>
<td>Chapman</td>
<td>Neil</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>ME2</td>
<td>35090</td>
<td>29JAN72</td>
<td>10JAN95</td>
<td>718-384-5618</td>
</tr>
<tr>
<td>1876</td>
<td>Chin</td>
<td>Jack</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>TA3</td>
<td>39675</td>
<td>23MAY66</td>
<td>30APR96</td>
<td>212-588-5634</td>
</tr>
<tr>
<td>1037</td>
<td>Chow</td>
<td>Jane</td>
<td>Stamford</td>
<td>CT</td>
<td>M</td>
<td>TA3</td>
<td>39675</td>
<td>23MAY66</td>
<td>30APR96</td>
<td>212-588-5634</td>
</tr>
<tr>
<td>IdNumber</td>
<td>LastName</td>
<td>FirstName</td>
<td>City</td>
<td>State</td>
<td>Gender</td>
<td>JobCode</td>
<td>Salary</td>
<td>Birth date</td>
<td>Hired date</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>---------</td>
<td>--------</td>
<td>------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>1129</td>
<td>Cook</td>
<td>Brenda</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>ME2</td>
<td>34929</td>
<td>11DEC79</td>
<td>20AUG03</td>
<td></td>
</tr>
<tr>
<td>1988</td>
<td>Cooper</td>
<td>Anthony</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>PA3</td>
<td>32217</td>
<td>03DEC57</td>
<td>21SEP92</td>
<td></td>
</tr>
<tr>
<td>1405</td>
<td>Davidson</td>
<td>Jason</td>
<td>Paterson</td>
<td>NJ</td>
<td>M</td>
<td>SCP</td>
<td>18056</td>
<td>08MAR54</td>
<td>29JAN00</td>
<td></td>
</tr>
<tr>
<td>1430</td>
<td>Dean</td>
<td>Brenda</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>ME2</td>
<td>32925</td>
<td>03MAR70</td>
<td>30APR05</td>
<td></td>
</tr>
<tr>
<td>1983</td>
<td>Cooper</td>
<td>Anthony</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>PA3</td>
<td>33419</td>
<td>03MAR50</td>
<td>30APR85</td>
<td></td>
</tr>
<tr>
<td>1134</td>
<td>Delgado</td>
<td>Maria</td>
<td>Stamford</td>
<td>CT</td>
<td>M</td>
<td>TA2</td>
<td>31426</td>
<td>08MAR77</td>
<td>24DEC04</td>
<td></td>
</tr>
<tr>
<td>1118</td>
<td>Dennis</td>
<td>Roger</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>ME2</td>
<td>35139</td>
<td>11JAN57</td>
<td>19DEC58</td>
<td></td>
</tr>
<tr>
<td>1438</td>
<td>Donaldson</td>
<td>Karen</td>
<td>Stamford</td>
<td>CT</td>
<td>F</td>
<td>TA2</td>
<td>39223</td>
<td>18MAR63</td>
<td>21NOV03</td>
<td></td>
</tr>
<tr>
<td>1125</td>
<td>Dunlap</td>
<td>Donna</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>PA2</td>
<td>28888</td>
<td>11NOV76</td>
<td>14DEC95</td>
<td></td>
</tr>
<tr>
<td>1475</td>
<td>Eaton</td>
<td>Alicia</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>PA2</td>
<td>27787</td>
<td>16DEC71</td>
<td>16JUL98</td>
<td></td>
</tr>
<tr>
<td>1117</td>
<td>Edgerton</td>
<td>Joshua</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>TA3</td>
<td>39771</td>
<td>08JUN56</td>
<td>16AUG00</td>
<td></td>
</tr>
<tr>
<td>1935</td>
<td>Fernandez</td>
<td>Katrina</td>
<td>Bridgeport CT</td>
<td>F</td>
<td>F</td>
<td>TA2</td>
<td>39223</td>
<td>36MAY63</td>
<td>21NOV03</td>
<td></td>
</tr>
<tr>
<td>1124</td>
<td>Fields</td>
<td>Diana</td>
<td>White Plains NY</td>
<td>F</td>
<td>F</td>
<td>PA2</td>
<td>28888</td>
<td>16MAR63</td>
<td>21NOV03</td>
<td></td>
</tr>
<tr>
<td>1422</td>
<td>Fletcher</td>
<td>Marie</td>
<td>Princeton NJ</td>
<td>F</td>
<td>F</td>
<td>PA2</td>
<td>27787</td>
<td>16DEC71</td>
<td>16JUL98</td>
<td></td>
</tr>
<tr>
<td>1616</td>
<td>Flowers</td>
<td>Annette</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>TA2</td>
<td>34137</td>
<td>04MAR68</td>
<td>07JUN01</td>
<td></td>
</tr>
<tr>
<td>1406</td>
<td>Foster</td>
<td>Gerald</td>
<td>Bridgeport CT</td>
<td>F</td>
<td>M</td>
<td>ME2</td>
<td>35185</td>
<td>11MAR69</td>
<td>20FEB95</td>
<td></td>
</tr>
<tr>
<td>1120</td>
<td>Garcia</td>
<td>Jack</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>ME1</td>
<td>28619</td>
<td>14SEP80</td>
<td>10OCT90</td>
<td></td>
</tr>
<tr>
<td>1094</td>
<td>Gomez</td>
<td>Alan</td>
<td>Bridgeport CT</td>
<td>M</td>
<td>M</td>
<td>PA1</td>
<td>22268</td>
<td>05APR78</td>
<td>20APR99</td>
<td></td>
</tr>
<tr>
<td>1389</td>
<td>Gordon</td>
<td>Levi</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>BCK</td>
<td>25028</td>
<td>21JUL67</td>
<td>21AUG03</td>
<td></td>
</tr>
<tr>
<td>1905</td>
<td>Graham</td>
<td>Alvin</td>
<td>New York</td>
<td>NY</td>
<td>M</td>
<td>PT1</td>
<td>65111</td>
<td>19APR80</td>
<td>01JUN98</td>
<td></td>
</tr>
<tr>
<td>1407</td>
<td>Grant</td>
<td>Daniel</td>
<td>Mt. Vernon NY</td>
<td>M</td>
<td>M</td>
<td>PT1</td>
<td>68096</td>
<td>26MAR77</td>
<td>21MAR98</td>
<td></td>
</tr>
<tr>
<td>1114</td>
<td>Green</td>
<td>Janice</td>
<td>New York</td>
<td>NY</td>
<td>F</td>
<td>TA2</td>
<td>32928</td>
<td>21SEP77</td>
<td>30JUN06</td>
<td></td>
</tr>
</tbody>
</table>

```
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.
    Hired date7. HomePhone $ 83-95;
  format birth hired date7.;
datalines;
1919 Adams Gerald Stamford CT M TA2 34376 15SEP70 07JUN05 203/781-1255
<table>
<thead>
<tr>
<th>Name</th>
<th>Last Name</th>
<th>First Name</th>
<th>City</th>
<th>State</th>
<th>Gender</th>
<th>Age</th>
<th>Birthdate</th>
<th>Deathdate</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexander</td>
<td>Susan</td>
<td>Bridgeport</td>
<td>CT</td>
<td>ME2</td>
<td>F</td>
<td>35108</td>
<td>18OCT72</td>
<td>12AUG98</td>
<td>203/675-7715</td>
</tr>
<tr>
<td>Apple</td>
<td>Troy</td>
<td>New York</td>
<td>NY</td>
<td>ME1</td>
<td>M</td>
<td>29769</td>
<td>08NOV85</td>
<td>19OCT06</td>
<td>212/586-0808</td>
</tr>
<tr>
<td>Arthur</td>
<td>Barbara</td>
<td>New York</td>
<td>NY</td>
<td>FA3</td>
<td>F</td>
<td>32886</td>
<td>03SEP63</td>
<td>01AUG00</td>
<td>718/383-1549</td>
</tr>
<tr>
<td>Avery</td>
<td>Jerry</td>
<td>Paterson</td>
<td>NJ</td>
<td>TA3</td>
<td>M</td>
<td>38822</td>
<td>16DEC68</td>
<td>20NOV93</td>
<td>201/732-8787</td>
</tr>
<tr>
<td>Barefoot</td>
<td>Joseph</td>
<td>Princeton</td>
<td>NJ</td>
<td>ME3</td>
<td>M</td>
<td>43025</td>
<td>29APR62</td>
<td>10JUN95</td>
<td>201/812-5665</td>
</tr>
<tr>
<td>Baucom</td>
<td>Walter</td>
<td>New York</td>
<td>NY</td>
<td>SCP</td>
<td>M</td>
<td>18723</td>
<td>09JUN80</td>
<td>04OCT98</td>
<td>212/586-8060</td>
</tr>
<tr>
<td>Blair</td>
<td>Blair</td>
<td>Stamford</td>
<td>CT</td>
<td>PT2</td>
<td>F</td>
<td>86060</td>
<td>02APR79</td>
<td>13FEB03</td>
<td>203/781-1777</td>
</tr>
<tr>
<td>Blalock</td>
<td>Avery</td>
<td>Paterson</td>
<td>NJ</td>
<td>ME1</td>
<td>M</td>
<td>32615</td>
<td>20JAN71</td>
<td>05DEC98</td>
<td>718/383-2849</td>
</tr>
<tr>
<td>Brandt</td>
<td>Marie</td>
<td>New York</td>
<td>NY</td>
<td>SCP</td>
<td>F</td>
<td>38785</td>
<td>25DEC66</td>
<td>08OCT03</td>
<td>718/384-8816</td>
</tr>
<tr>
<td>Bowden</td>
<td>Earl</td>
<td>Bridgeport</td>
<td>CT</td>
<td>ME1</td>
<td>M</td>
<td>28072</td>
<td>31JAN79</td>
<td>24DEC99</td>
<td>201/732-8787</td>
</tr>
<tr>
<td>Boyce</td>
<td>Jonathan</td>
<td>New York</td>
<td>NY</td>
<td>TP1</td>
<td>F</td>
<td>66517</td>
<td>28DEC82</td>
<td>30JAN00</td>
<td>212/587-1247</td>
</tr>
<tr>
<td>Boyce</td>
<td>Marie</td>
<td>New York</td>
<td>NY</td>
<td>TA2</td>
<td>M</td>
<td>32615</td>
<td>20JAN71</td>
<td>05DEC98</td>
<td>718/384-8816</td>
</tr>
<tr>
<td>Bowden</td>
<td>Jerry</td>
<td>New York</td>
<td>NY</td>
<td>TA3</td>
<td>F</td>
<td>38785</td>
<td>25DEC66</td>
<td>08OCT03</td>
<td>718/384-8816</td>
</tr>
<tr>
<td>Bowden</td>
<td>Joseph</td>
<td>Princeton</td>
<td>NJ</td>
<td>ME3</td>
<td>M</td>
<td>43025</td>
<td>29APR62</td>
<td>10JUN95</td>
<td>201/812-5665</td>
</tr>
<tr>
<td>Bowden</td>
<td>Joseph</td>
<td>Princeton</td>
<td>NJ</td>
<td>ME3</td>
<td>M</td>
<td>43025</td>
<td>29APR62</td>
<td>10JUN95</td>
<td>201/812-5665</td>
</tr>
<tr>
<td>Bowden</td>
<td>Joseph</td>
<td>Princeton</td>
<td>NJ</td>
<td>ME3</td>
<td>M</td>
<td>43025</td>
<td>29APR62</td>
<td>10JUN95</td>
<td>201/812-5665</td>
</tr>
</tbody>
</table>

**ENERGY**

```sas
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.

```sas
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;
```
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      1/6/16            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     36.90
Montserrat              120127       1/2/16      1/2/16            In Store   19    184.20      9.25
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    5    41.40      18.00
St. Helena              120360       1/2/16      1/2/16            Internet    9    94.70      47.45
St. Pierre/Miquelon     120842       1/2/16      1/6/16            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            In Store   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

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
;
```sql
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
```
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              -10
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              -10
202     03MAR08  LGA   ORD   No Delay         Domestic               -1
219     03MAR08  LGA   LON   1-10 Minutes     International           4
622     03MAR08  LGA   FRA   No Delay         International           0
132     03MAR08  LGA   YYZ   1-10 Minutes     International           6
271     03MAR08  LGA   PAR   1-10 Minutes     International           2
;
<table>
<thead>
<tr>
<th>ID</th>
<th>Date</th>
<th>Origin</th>
<th>Destination</th>
<th>Delay Code</th>
<th>Flight Type</th>
<th>Delay Time</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>302</td>
<td>03MAR08</td>
<td>LGA</td>
<td>WAS</td>
<td>1-10 Minutes</td>
<td>Domestic</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>04MAR08</td>
<td>LGA</td>
<td>LAX</td>
<td>11+ Minutes</td>
<td>Domestic</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>04MAR08</td>
<td>LGA</td>
<td>ORD</td>
<td>No Delay</td>
<td>Domestic</td>
<td>-5</td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>04MAR08</td>
<td>LGA</td>
<td>LON</td>
<td>1-10 Minutes</td>
<td>International</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>622</td>
<td>04MAR08</td>
<td>LGA</td>
<td>FRA</td>
<td>11+ Minutes</td>
<td>International</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>04MAR08</td>
<td>LGA</td>
<td>YYZ</td>
<td>No Delay</td>
<td>International</td>
<td>-5</td>
<td></td>
</tr>
<tr>
<td>271</td>
<td>04MAR08</td>
<td>LGA</td>
<td>PAR</td>
<td>1-10 Minutes</td>
<td>International</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>04MAR08</td>
<td>LGA</td>
<td>WAS</td>
<td>1-10 Minutes</td>
<td>Domestic</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>05MAR08</td>
<td>LGA</td>
<td>LAX</td>
<td>No Delay</td>
<td>Domestic</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>05MAR08</td>
<td>LGA</td>
<td>ORD</td>
<td>1-10 Minutes</td>
<td>Domestic</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>05MAR08</td>
<td>LGA</td>
<td>LON</td>
<td>1-10 Minutes</td>
<td>International</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>622</td>
<td>05MAR08</td>
<td>LGA</td>
<td>FRA</td>
<td>No Delay</td>
<td>International</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>05MAR08</td>
<td>LGA</td>
<td>YYZ</td>
<td>1-10 Minutes</td>
<td>International</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>271</td>
<td>05MAR08</td>
<td>LGA</td>
<td>PAR</td>
<td>1-10 Minutes</td>
<td>International</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>06MAR08</td>
<td>LGA</td>
<td>LAX</td>
<td>No Delay</td>
<td>Domestic</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>06MAR08</td>
<td>LGA</td>
<td>ORD</td>
<td>No Delay</td>
<td>Domestic</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>06MAR08</td>
<td>LGA</td>
<td>LON</td>
<td>11+ Minutes</td>
<td>International</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>06MAR08</td>
<td>LGA</td>
<td>YYZ</td>
<td>1-10 Minutes</td>
<td>International</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>06MAR08</td>
<td>LGA</td>
<td>WAS</td>
<td>1-10 Minutes</td>
<td>Domestic</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>07MAR08</td>
<td>LGA</td>
<td>LAX</td>
<td>No Delay</td>
<td>Domestic</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>07MAR08</td>
<td>LGA</td>
<td>ORD</td>
<td>No Delay</td>
<td>Domestic</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>07MAR08</td>
<td>LGA</td>
<td>LON</td>
<td>11+ Minutes</td>
<td>International</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>622</td>
<td>07MAR08</td>
<td>LGA</td>
<td>FRA</td>
<td>11+ Minutes</td>
<td>International</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>07MAR08</td>
<td>LGA</td>
<td>YYZ</td>
<td>No Delay</td>
<td>International</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>271</td>
<td>07MAR08</td>
<td>LGA</td>
<td>PAR</td>
<td>1-10 Minutes</td>
<td>International</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>07MAR08</td>
<td>LGA</td>
<td>WAS</td>
<td>No Delay</td>
<td>Domestic</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

---

**PROCLIB.EMP95**

```plaintext
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
;
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
;
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  0.03  
NW Falls    0.01  0.02  0.59  0.58  0.35  0.48  
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  0.12  
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  

This data set (table) is updated in “Updating Data in a PROC SQL Table” in SAS Viya SQL Procedure User’s Guide, and its updated data is used in subsequent examples.

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 hire7. 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  26AUG63  10FEB92
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  17AUG91
1269  M  NA1  41690  03MAY72  28NOV92
1065  M  ME2  35090  26JAN44  07JAN87
1876  M  TA3  39675  20MAI58  27APR85
1037  F  TA1  28558  10APR64  13SEP92
1129  F  ME2  34929  08DEC61  17AUG91
1988  M  FA3  32217  30NOV59  18SEP84
<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Gender</th>
<th>Code</th>
<th>Name</th>
<th>Date of Birth</th>
<th>Date of Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1405</td>
<td>M</td>
<td>SCP</td>
<td></td>
<td>18056</td>
<td>05MAR66</td>
</tr>
<tr>
<td>1430</td>
<td>F</td>
<td>TA2</td>
<td></td>
<td>32925</td>
<td>28FEB62</td>
</tr>
<tr>
<td>1983</td>
<td>F</td>
<td>FA3</td>
<td></td>
<td>33419</td>
<td>28FEB62</td>
</tr>
<tr>
<td>1134</td>
<td>F</td>
<td>TA2</td>
<td></td>
<td>33462</td>
<td>05MAR69</td>
</tr>
<tr>
<td>1118</td>
<td>M</td>
<td>PT3</td>
<td></td>
<td>111379</td>
<td>05MAR69</td>
</tr>
<tr>
<td>1438</td>
<td>F</td>
<td>TA3</td>
<td></td>
<td>39223</td>
<td>05MAR66</td>
</tr>
<tr>
<td>1125</td>
<td>F</td>
<td>FA2</td>
<td></td>
<td>28888</td>
<td>05MAR66</td>
</tr>
<tr>
<td>1475</td>
<td>F</td>
<td>FA2</td>
<td></td>
<td>27787</td>
<td>15DEC61</td>
</tr>
<tr>
<td>1117</td>
<td>M</td>
<td>TA3</td>
<td></td>
<td>39771</td>
<td>05JUN63</td>
</tr>
<tr>
<td>1935</td>
<td>F</td>
<td>NA2</td>
<td></td>
<td>51081</td>
<td>15DEC61</td>
</tr>
<tr>
<td>1124</td>
<td>F</td>
<td>FA1</td>
<td></td>
<td>23177</td>
<td>10JUL58</td>
</tr>
<tr>
<td>1422</td>
<td>F</td>
<td>FA1</td>
<td></td>
<td>22454</td>
<td>04JUN63</td>
</tr>
<tr>
<td>1616</td>
<td>F</td>
<td>TA2</td>
<td></td>
<td>34137</td>
<td>01MAR70</td>
</tr>
<tr>
<td>1406</td>
<td>M</td>
<td>ME2</td>
<td></td>
<td>35185</td>
<td>08MAR61</td>
</tr>
<tr>
<td>1120</td>
<td>M</td>
<td>ME1</td>
<td></td>
<td>28619</td>
<td>11SEP72</td>
</tr>
<tr>
<td>1094</td>
<td>M</td>
<td>FA1</td>
<td></td>
<td>22268</td>
<td>02APR70</td>
</tr>
<tr>
<td>1389</td>
<td>M</td>
<td>BCK</td>
<td></td>
<td>25028</td>
<td>15DEC61</td>
</tr>
<tr>
<td>1905</td>
<td>M</td>
<td>PT1</td>
<td></td>
<td>65111</td>
<td>16APR72</td>
</tr>
<tr>
<td>1407</td>
<td>M</td>
<td>PT1</td>
<td></td>
<td>68096</td>
<td>15DEC61</td>
</tr>
<tr>
<td>1114</td>
<td>F</td>
<td>TA2</td>
<td></td>
<td>32928</td>
<td>18SEP69</td>
</tr>
<tr>
<td>1410</td>
<td>M</td>
<td>PT2</td>
<td></td>
<td>84685</td>
<td>03MAR67</td>
</tr>
<tr>
<td>1439</td>
<td>F</td>
<td>PT1</td>
<td></td>
<td>70736</td>
<td>06MAR64</td>
</tr>
<tr>
<td>1409</td>
<td>M</td>
<td>ME3</td>
<td></td>
<td>41551</td>
<td>19APR50</td>
</tr>
<tr>
<td>1408</td>
<td>M</td>
<td>TA2</td>
<td></td>
<td>34138</td>
<td>28SEP62</td>
</tr>
<tr>
<td>1121</td>
<td>M</td>
<td>ME1</td>
<td></td>
<td>29112</td>
<td>26SEP71</td>
</tr>
<tr>
<td>1991</td>
<td>F</td>
<td>TA1</td>
<td></td>
<td>27645</td>
<td>07MAR61</td>
</tr>
<tr>
<td>1102</td>
<td>M</td>
<td>TA2</td>
<td></td>
<td>34542</td>
<td>01OCT59</td>
</tr>
<tr>
<td>1356</td>
<td>M</td>
<td>ME2</td>
<td></td>
<td>36869</td>
<td>26SEP57</td>
</tr>
<tr>
<td>1545</td>
<td>M</td>
<td>PT1</td>
<td></td>
<td>66130</td>
<td>12AUG59</td>
</tr>
<tr>
<td>1292</td>
<td>F</td>
<td>ME2</td>
<td></td>
<td>36691</td>
<td>28OCT64</td>
</tr>
<tr>
<td>1440</td>
<td>F</td>
<td>ME2</td>
<td></td>
<td>35785</td>
<td>24SEP62</td>
</tr>
<tr>
<td>1368</td>
<td>M</td>
<td>FA2</td>
<td></td>
<td>27808</td>
<td>11JUN61</td>
</tr>
<tr>
<td>1369</td>
<td>M</td>
<td>TA2</td>
<td></td>
<td>33705</td>
<td>28DEC61</td>
</tr>
<tr>
<td>1411</td>
<td>M</td>
<td>FA2</td>
<td></td>
<td>27265</td>
<td>27MAR61</td>
</tr>
<tr>
<td>1113</td>
<td>F</td>
<td>FA1</td>
<td></td>
<td>22367</td>
<td>15JAN68</td>
</tr>
<tr>
<td>1704</td>
<td>M</td>
<td>BCK</td>
<td></td>
<td>25465</td>
<td>30AUG66</td>
</tr>
<tr>
<td>1900</td>
<td>M</td>
<td>ME2</td>
<td></td>
<td>35105</td>
<td>25MAR62</td>
</tr>
<tr>
<td>1126</td>
<td>F</td>
<td>TA3</td>
<td></td>
<td>40899</td>
<td>28MAR63</td>
</tr>
<tr>
<td>1677</td>
<td>M</td>
<td>BCK</td>
<td></td>
<td>26007</td>
<td>05NOV63</td>
</tr>
<tr>
<td>1441</td>
<td>F</td>
<td>FA2</td>
<td></td>
<td>27158</td>
<td>19NOV69</td>
</tr>
<tr>
<td>1421</td>
<td>M</td>
<td>TA2</td>
<td></td>
<td>33155</td>
<td>08JAN59</td>
</tr>
<tr>
<td>1119</td>
<td>M</td>
<td>TA1</td>
<td></td>
<td>26924</td>
<td>20JUN62</td>
</tr>
<tr>
<td>1834</td>
<td>M</td>
<td>BCK</td>
<td></td>
<td>26896</td>
<td>08FEB72</td>
</tr>
<tr>
<td>1777</td>
<td>M</td>
<td>PT3</td>
<td></td>
<td>109630</td>
<td>23MAR51</td>
</tr>
<tr>
<td>1663</td>
<td>M</td>
<td>BCK</td>
<td></td>
<td>26452</td>
<td>11JAN67</td>
</tr>
<tr>
<td>1106</td>
<td>M</td>
<td>PT2</td>
<td></td>
<td>89632</td>
<td>06NOV57</td>
</tr>
<tr>
<td>1103</td>
<td>F</td>
<td>FA1</td>
<td></td>
<td>23738</td>
<td>16FEB68</td>
</tr>
<tr>
<td>1477</td>
<td>M</td>
<td>FA2</td>
<td></td>
<td>28566</td>
<td>21MAR64</td>
</tr>
<tr>
<td>1476</td>
<td>F</td>
<td>TA2</td>
<td></td>
<td>34803</td>
<td>30MAR64</td>
</tr>
<tr>
<td>1379</td>
<td>M</td>
<td>ME3</td>
<td></td>
<td>42264</td>
<td>08AUG61</td>
</tr>
<tr>
<td>1104</td>
<td>M</td>
<td>SCP</td>
<td></td>
<td>17946</td>
<td>25MAR63</td>
</tr>
<tr>
<td>1009</td>
<td>M</td>
<td>TA1</td>
<td></td>
<td>28880</td>
<td>02MAR59</td>
</tr>
<tr>
<td>1412</td>
<td>M</td>
<td>ME1</td>
<td></td>
<td>27799</td>
<td>18JUN56</td>
</tr>
<tr>
<td>1115</td>
<td>F</td>
<td>FA3</td>
<td></td>
<td>32699</td>
<td>22AUG60</td>
</tr>
<tr>
<td>1128</td>
<td>F</td>
<td>TA2</td>
<td></td>
<td>32777</td>
<td>23MAR65</td>
</tr>
<tr>
<td>1442</td>
<td>F</td>
<td>PT2</td>
<td></td>
<td>84536</td>
<td>05SEP66</td>
</tr>
<tr>
<td>ID</td>
<td>Sex</td>
<td>Code</td>
<td>Date</td>
<td>Age</td>
<td>Month</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>1417</td>
<td>M</td>
<td>NA2</td>
<td>52270</td>
<td>27JUN64</td>
<td>07MAR89</td>
</tr>
<tr>
<td>1478</td>
<td>M</td>
<td>PT2</td>
<td>84203</td>
<td>09AUG59</td>
<td>24OCT90</td>
</tr>
<tr>
<td>1673</td>
<td>M</td>
<td>BCK</td>
<td>25477</td>
<td>27FEB70</td>
<td>15JUL91</td>
</tr>
<tr>
<td>1839</td>
<td>F</td>
<td>NA1</td>
<td>43433</td>
<td>29NOV70</td>
<td>03JUL93</td>
</tr>
<tr>
<td>1347</td>
<td>M</td>
<td>TA3</td>
<td>40079</td>
<td>21SEP70</td>
<td>06SEP84</td>
</tr>
<tr>
<td>1423</td>
<td>F</td>
<td>ME2</td>
<td>35773</td>
<td>14MAY68</td>
<td>19AUG90</td>
</tr>
<tr>
<td>1200</td>
<td>F</td>
<td>ME1</td>
<td>27816</td>
<td>10JAN71</td>
<td>14AUG92</td>
</tr>
<tr>
<td>1970</td>
<td>F</td>
<td>FA1</td>
<td>22615</td>
<td>25SEP64</td>
<td>02JAN90</td>
</tr>
<tr>
<td>1521</td>
<td>M</td>
<td>ME3</td>
<td>41526</td>
<td>12APR63</td>
<td>13JUL88</td>
</tr>
<tr>
<td>1354</td>
<td>F</td>
<td>SCP</td>
<td>18335</td>
<td>29MAY71</td>
<td>16JUN92</td>
</tr>
<tr>
<td>1424</td>
<td>F</td>
<td>FA2</td>
<td>28978</td>
<td>04AUG69</td>
<td>11DEC88</td>
</tr>
<tr>
<td>1132</td>
<td>F</td>
<td>FA1</td>
<td>22413</td>
<td>30MAY72</td>
<td>22OCT93</td>
</tr>
<tr>
<td>1945</td>
<td>M</td>
<td>BCK</td>
<td>25996</td>
<td>20NOV59</td>
<td>14MAR80</td>
</tr>
<tr>
<td>1556</td>
<td>M</td>
<td>PT1</td>
<td>71349</td>
<td>22JUN64</td>
<td>11DEC91</td>
</tr>
<tr>
<td>1413</td>
<td>M</td>
<td>FA2</td>
<td>27435</td>
<td>16SEP65</td>
<td>02JAN90</td>
</tr>
<tr>
<td>1352</td>
<td>M</td>
<td>NA2</td>
<td>53798</td>
<td>02DEC60</td>
<td>16OCT96</td>
</tr>
<tr>
<td>1555</td>
<td>F</td>
<td>FA2</td>
<td>27499</td>
<td>16MAR68</td>
<td>04JUL92</td>
</tr>
<tr>
<td>1038</td>
<td>F</td>
<td>TA1</td>
<td>26533</td>
<td>09NOV69</td>
<td>23NOV91</td>
</tr>
<tr>
<td>1420</td>
<td>M</td>
<td>ME3</td>
<td>43071</td>
<td>19FEB65</td>
<td>22JUL87</td>
</tr>
<tr>
<td>1561</td>
<td>M</td>
<td>TA2</td>
<td>34514</td>
<td>30NOV63</td>
<td>07OCT87</td>
</tr>
<tr>
<td>1434</td>
<td>F</td>
<td>FA2</td>
<td>28622</td>
<td>11JUL62</td>
<td>28OCT90</td>
</tr>
<tr>
<td>1414</td>
<td>M</td>
<td>FA1</td>
<td>23644</td>
<td>24MAR72</td>
<td>12APR92</td>
</tr>
<tr>
<td>1112</td>
<td>M</td>
<td>TA1</td>
<td>26905</td>
<td>29NOV64</td>
<td>07DEC92</td>
</tr>
<tr>
<td>1390</td>
<td>M</td>
<td>FA2</td>
<td>27761</td>
<td>19FEB65</td>
<td>23JUN91</td>
</tr>
<tr>
<td>1332</td>
<td>M</td>
<td>NA1</td>
<td>42178</td>
<td>17SEP70</td>
<td>04JUN91</td>
</tr>
<tr>
<td>1890</td>
<td>M</td>
<td>PT2</td>
<td>91908</td>
<td>20JUL51</td>
<td>25NOV79</td>
</tr>
<tr>
<td>1429</td>
<td>F</td>
<td>TA1</td>
<td>27939</td>
<td>28FEB60</td>
<td>07AUG92</td>
</tr>
<tr>
<td>1107</td>
<td>M</td>
<td>PT2</td>
<td>89977</td>
<td>09JUN54</td>
<td>10FEB79</td>
</tr>
<tr>
<td>1908</td>
<td>F</td>
<td>TA2</td>
<td>32995</td>
<td>10DEC69</td>
<td>23APR90</td>
</tr>
<tr>
<td>1830</td>
<td>F</td>
<td>PT2</td>
<td>84471</td>
<td>27MAY57</td>
<td>29JAN83</td>
</tr>
<tr>
<td>1882</td>
<td>M</td>
<td>ME3</td>
<td>41538</td>
<td>10JUL57</td>
<td>21NOV78</td>
</tr>
<tr>
<td>1050</td>
<td>M</td>
<td>ME2</td>
<td>35167</td>
<td>14JUL63</td>
<td>24AUG86</td>
</tr>
<tr>
<td>1425</td>
<td>F</td>
<td>FA1</td>
<td>21979</td>
<td>28DEC71</td>
<td>28FEB93</td>
</tr>
<tr>
<td>1928</td>
<td>M</td>
<td>PT2</td>
<td>89858</td>
<td>16SEP54</td>
<td>13JUL90</td>
</tr>
<tr>
<td>1480</td>
<td>F</td>
<td>TA3</td>
<td>39583</td>
<td>03SEP57</td>
<td>25MAR81</td>
</tr>
<tr>
<td>1100</td>
<td>M</td>
<td>BCK</td>
<td>25004</td>
<td>01DEC60</td>
<td>07MAY88</td>
</tr>
<tr>
<td>1995</td>
<td>F</td>
<td>ME1</td>
<td>28810</td>
<td>24AUG73</td>
<td>19SEP93</td>
</tr>
<tr>
<td>1135</td>
<td>F</td>
<td>FA2</td>
<td>27321</td>
<td>20SEP60</td>
<td>31MAR90</td>
</tr>
<tr>
<td>1415</td>
<td>M</td>
<td>FA2</td>
<td>28278</td>
<td>09MAR58</td>
<td>12FEB88</td>
</tr>
<tr>
<td>1076</td>
<td>M</td>
<td>PT1</td>
<td>66558</td>
<td>14OCT55</td>
<td>03OCT91</td>
</tr>
<tr>
<td>1426</td>
<td>F</td>
<td>TA2</td>
<td>32991</td>
<td>05DEC66</td>
<td>25JUN90</td>
</tr>
<tr>
<td>1564</td>
<td>F</td>
<td>SCP</td>
<td>18833</td>
<td>12APR62</td>
<td>01JUL92</td>
</tr>
<tr>
<td>1221</td>
<td>F</td>
<td>FA2</td>
<td>27896</td>
<td>22SEP67</td>
<td>04OCT91</td>
</tr>
<tr>
<td>1133</td>
<td>M</td>
<td>TA1</td>
<td>27701</td>
<td>13JUL66</td>
<td>12FEB92</td>
</tr>
<tr>
<td>1435</td>
<td>F</td>
<td>TA3</td>
<td>38808</td>
<td>12MAY59</td>
<td>08FEB80</td>
</tr>
<tr>
<td>1418</td>
<td>M</td>
<td>ME1</td>
<td>28005</td>
<td>29MAR57</td>
<td>06JAN92</td>
</tr>
<tr>
<td>1017</td>
<td>M</td>
<td>TA3</td>
<td>40858</td>
<td>28DEC57</td>
<td>16OCT81</td>
</tr>
<tr>
<td>1443</td>
<td>F</td>
<td>NA1</td>
<td>42274</td>
<td>17NOV68</td>
<td>29AUG91</td>
</tr>
<tr>
<td>1131</td>
<td>F</td>
<td>TA2</td>
<td>32575</td>
<td>26DEC71</td>
<td>19APR91</td>
</tr>
</tbody>
</table>
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     718/386-1549
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   BREW2ZAK         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     201/732-3323
1430   DABROWSKI        SANDRA           BRIDGEPORT       CT     203/675-1647
1983   DEAN             SHARON           NEW YORK         NY     718/384-1647
1134   DIBLIGADO        MARIA            STAMFORD         CT     203/781-1528
1118   DENNIS           ROGER            NEW YORK         NY     718/383-1122
1438   DABBOUSSI        KAMILLE          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 DANIEL MT. VERNON NY 914/468-1616
1114 HERMANDEZ ROBERTO NEW YORK NY 718/384-3313
1905 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 JOHNSON JACK PATERSON NJ 201/732-3678
1133 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 RAYMOND STAMFORD CT 203/781-3337
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
<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>First Name</th>
<th>Location</th>
<th>State</th>
<th>City</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1354</td>
<td>PARKER</td>
<td>MARY</td>
<td>WHITE PLAINS</td>
<td>NY</td>
<td></td>
<td>914/455-2337</td>
</tr>
<tr>
<td>1424</td>
<td>PATTERSON</td>
<td>RENEE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/587-8991</td>
</tr>
<tr>
<td>1132</td>
<td>PEARCE</td>
<td>CAROL</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-1986</td>
</tr>
<tr>
<td>1845</td>
<td>PEARSON</td>
<td>JAMES</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-2311</td>
</tr>
<tr>
<td>1556</td>
<td>PENNINGTON</td>
<td>MICHAEL</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-5681</td>
</tr>
<tr>
<td>1413</td>
<td>PETERS</td>
<td>RANDALL</td>
<td>PRINCETON</td>
<td>NJ</td>
<td></td>
<td>201/812-2478</td>
</tr>
<tr>
<td>1123</td>
<td>PETERSON</td>
<td>SUZANNE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-0077</td>
</tr>
<tr>
<td>1907</td>
<td>PHELPS</td>
<td>WILLIAM</td>
<td>STAMFORD</td>
<td>CT</td>
<td></td>
<td>203/781-1118</td>
</tr>
<tr>
<td>1436</td>
<td>PORTER</td>
<td>SUSAN</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-5777</td>
</tr>
<tr>
<td>1385</td>
<td>RAYNOR</td>
<td>MILTON</td>
<td>BRIDGEPORT</td>
<td>CT</td>
<td></td>
<td>203/675-2846</td>
</tr>
<tr>
<td>1432</td>
<td>REED</td>
<td>MARILYN</td>
<td>MT. VERNON</td>
<td>NY</td>
<td></td>
<td>914/468-5454</td>
</tr>
<tr>
<td>1111</td>
<td>RHODES</td>
<td>JEREMY</td>
<td>PRINCETON</td>
<td>NJ</td>
<td></td>
<td>201/812-1837</td>
</tr>
<tr>
<td>1116</td>
<td>RICHARDS</td>
<td>CASEY</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/587-1224</td>
</tr>
<tr>
<td>1352</td>
<td>RIVERS</td>
<td>SIMON</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-3345</td>
</tr>
<tr>
<td>1555</td>
<td>RODRIGUEZ</td>
<td>JULIA</td>
<td>BRIDGEPORT</td>
<td>CT</td>
<td></td>
<td>203/675-2401</td>
</tr>
<tr>
<td>1038</td>
<td>RODRIGUEZ</td>
<td>MARIA</td>
<td>BRIDGEPORT</td>
<td>CT</td>
<td></td>
<td>203/675-2048</td>
</tr>
<tr>
<td>1420</td>
<td>ROUSE</td>
<td>JEREMY</td>
<td>PATERNIA</td>
<td>NJ</td>
<td></td>
<td>201/732-9834</td>
</tr>
<tr>
<td>1561</td>
<td>SANDERS</td>
<td>RAYMOND</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/588-6615</td>
</tr>
<tr>
<td>1434</td>
<td>SANDERSON</td>
<td>EDITH</td>
<td>STAMFORD</td>
<td>CT</td>
<td></td>
<td>203/781-1333</td>
</tr>
<tr>
<td>1414</td>
<td>SANDERSON</td>
<td>NATHAN</td>
<td>BRIDGEPORT</td>
<td>CT</td>
<td></td>
<td>203/675-1715</td>
</tr>
<tr>
<td>1112</td>
<td>SANYERS</td>
<td>RANDY</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-4895</td>
</tr>
<tr>
<td>1390</td>
<td>SMART</td>
<td>JONATHAN</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-1141</td>
</tr>
<tr>
<td>1332</td>
<td>STEPHENSON</td>
<td>ADAM</td>
<td>BRIDGEPORT</td>
<td>CT</td>
<td></td>
<td>203/675-1497</td>
</tr>
<tr>
<td>1890</td>
<td>STEPHENSON</td>
<td>ROBERT</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-9874</td>
</tr>
<tr>
<td>1429</td>
<td>THOMPSON</td>
<td>ALICE</td>
<td>STAMFORD</td>
<td>CT</td>
<td></td>
<td>203/781-3857</td>
</tr>
<tr>
<td>1107</td>
<td>THOMPSON</td>
<td>WAYNE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-3785</td>
</tr>
<tr>
<td>1908</td>
<td>TRENTON</td>
<td>MELISSA</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/586-6262</td>
</tr>
<tr>
<td>1830</td>
<td>TRIPP</td>
<td>KATHY</td>
<td>BRIDGEPORT</td>
<td>CT</td>
<td></td>
<td>203/675-2479</td>
</tr>
<tr>
<td>1982</td>
<td>TUCKER</td>
<td>ALAN</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-0216</td>
</tr>
<tr>
<td>1050</td>
<td>TUTTLE</td>
<td>THOMAS</td>
<td>WHITE PLAINS</td>
<td>NY</td>
<td></td>
<td>914/455-2119</td>
</tr>
<tr>
<td>1425</td>
<td>UNDERWOOD</td>
<td>JENNY</td>
<td>STAMFORD</td>
<td>CT</td>
<td></td>
<td>203/781-0978</td>
</tr>
<tr>
<td>1928</td>
<td>UPCHURCH</td>
<td>LARRY</td>
<td>WHITE PLAINS</td>
<td>NY</td>
<td></td>
<td>914/455-5009</td>
</tr>
<tr>
<td>1480</td>
<td>UPDIKE</td>
<td>THERESA</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/587-8729</td>
</tr>
<tr>
<td>1100</td>
<td>VANDEUSEN</td>
<td>RICHARD</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/586-2531</td>
</tr>
<tr>
<td>1995</td>
<td>VARNER</td>
<td>ELIZABETH</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-7113</td>
</tr>
<tr>
<td>1135</td>
<td>VEGA</td>
<td>ANNA</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-5913</td>
</tr>
<tr>
<td>1415</td>
<td>VEGA</td>
<td>FRANKLIN</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-2823</td>
</tr>
<tr>
<td>1076</td>
<td>VENTER</td>
<td>RANDALL</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-2321</td>
</tr>
<tr>
<td>1426</td>
<td>VICK</td>
<td>THERESA</td>
<td>PRINCETON</td>
<td>NJ</td>
<td></td>
<td>201/812-2424</td>
</tr>
<tr>
<td>1564</td>
<td>WALTERS</td>
<td>ANNE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/587-3257</td>
</tr>
<tr>
<td>1221</td>
<td>WALTERS</td>
<td>DIANE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-1918</td>
</tr>
<tr>
<td>1133</td>
<td>WANG</td>
<td>CHIN</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/587-1956</td>
</tr>
<tr>
<td>1435</td>
<td>WARD</td>
<td>ELAINE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-4987</td>
</tr>
<tr>
<td>1418</td>
<td>WATSON</td>
<td>BERNARD</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-1298</td>
</tr>
<tr>
<td>1017</td>
<td>WELCH</td>
<td>DARIUS</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/586-5535</td>
</tr>
<tr>
<td>1443</td>
<td>WELLS</td>
<td>AGNES</td>
<td>STAMFORD</td>
<td>CT</td>
<td></td>
<td>203/781-5546</td>
</tr>
<tr>
<td>1131</td>
<td>WELLS</td>
<td>NADINE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/383-1045</td>
</tr>
<tr>
<td>1427</td>
<td>WHALEY</td>
<td>CAROLYN</td>
<td>MT. VERNON</td>
<td>NY</td>
<td></td>
<td>914/468-4528</td>
</tr>
<tr>
<td>1036</td>
<td>WONG</td>
<td>LESLIE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/587-2570</td>
</tr>
<tr>
<td>1130</td>
<td>WOOD</td>
<td>DEBORAH</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/587-0013</td>
</tr>
<tr>
<td>1127</td>
<td>WOOD</td>
<td>SANDRA</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>212/587-2881</td>
</tr>
<tr>
<td>1433</td>
<td>YANCEY</td>
<td>ROBIN</td>
<td>PRINCETON</td>
<td>NJ</td>
<td></td>
<td>201/812-1874</td>
</tr>
<tr>
<td>1431</td>
<td>YOUNG</td>
<td>DEBORAH</td>
<td>STAMFORD</td>
<td>CT</td>
<td></td>
<td>203/781-2987</td>
</tr>
<tr>
<td>1122</td>
<td>YOUNG</td>
<td>JOANN</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-2021</td>
</tr>
<tr>
<td>1105</td>
<td>YOUNG</td>
<td>LAWRENCE</td>
<td>NEW YORK</td>
<td>NY</td>
<td></td>
<td>718/384-0008</td>
</tr>
</tbody>
</table>


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 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
833 28 f 3 4 1
1 0 0 0 1 1 1 1 0 0 0 1 1
677 24 f 3 1 2
2 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
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 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
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 8
781 34 f 9 3 1
2 1 0 1 4 4 4 0 1 1 1 1 4 4
467 30 f 1 4 1 0 0 0 0 1 0 0 6 0 0 1 1 1 0 6 859 32 f 3 5 2 2 2 2 2 2 6 6 2 2 2 2 6 851 43 f 8 1 5 7 5 5 0 0 0 4 0 0 0 0 0 848 29 f 3 5 1 7 0 0 0 7 1 0 0 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 1 0 2 1 1 1 0 1 0 0 0 3 1 1 0 0 0 222 23 f 3 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 0 859 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 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 5 0 0 0 0 7 0 0 0 0 0 221 27 m 1 1 7 7 0 0 0 0 0 0 5 0 0 0 5 0 684 19 f 0 .5 1 7 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 1 0 0 5 5 5 5 1 0 0 684 18 m 1 .5 1 0 2 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 1 0 0 0 221 29 m .5 .5 5 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 8 0 0 0 966 23 f 1 2 1 1 5 5 5 1 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 5 0 0 1 0 0 0 1
<p>| | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>783</td>
<td>29</td>
<td>f</td>
<td>1.5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>722</td>
<td>29</td>
<td>f</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>856</td>
<td>30</td>
<td>f</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>722</td>
<td>30</td>
<td>f</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>546</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>382</td>
<td>40</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>849</td>
<td>32</td>
<td>m</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>542</td>
<td>36</td>
<td>f</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>546</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>722</td>
<td>30</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>966</td>
<td>32</td>
<td>f</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>851</td>
<td>40</td>
<td>m</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>783</td>
<td>29</td>
<td>f</td>
<td>1.5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>722</td>
<td>28</td>
<td>f</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>856</td>
<td>45</td>
<td>f</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>849</td>
<td>32</td>
<td>m</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>546</td>
<td>30</td>
<td>f</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>30</td>
<td>f</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>222</td>
<td>35</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>467</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>542</td>
<td>36</td>
<td>f</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>382</td>
<td>40</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>849</td>
<td>32</td>
<td>m</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>542</td>
<td>36</td>
<td>f</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>546</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>222</td>
<td>35</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>467</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>222</td>
<td>35</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>467</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>222</td>
<td>35</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>467</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>222</td>
<td>35</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>467</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>222</td>
<td>35</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>467</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>222</td>
<td>35</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>467</td>
<td>30</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>677</td>
<td>33</td>
<td>m</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>490</td>
<td>27</td>
<td>f</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>222</td>
<td>35</td>
<td>f</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
859 27 f 2 5 2
6 0 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 2 0 8 0 2 0
967 38 f 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 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 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 8 8 0 0 8 8 8 0 0
688 26 m 3 1 1
1 1 7 1 1 7 0 0 8 8 0 0 0
490 41 m 2 2 5
5 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 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 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 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 m 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 8 8 8 0 0 0
688 31 m 8 2 5
7 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 0
677 37 f 3 3 1
0 0 1 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 0 2
781 30 f 10 4 2
2 0 0 2 0 2 0 0 0 0 0 0 0 0 0 2
362 27 m 12 4 3
3 1 1 1 3 3 3 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
779 37 f 6 3 1
1 1 1 1 0 0 0 1 0 0 0 1 0
467 32 f 1 1 2
2 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
779 28 m 5 2 1
1 1 1 1 0 0 0 0 0 7 7 1 1 0
677 28 m 3 1 5
7 5 5 5 6 0 0 6 6 6 6 6 0
677 25 f 9 2 5
1 5 5 5 1 1 0 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 0 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
851 43 f 8 1 6
4 6 0 6 4 0 0 0 0 0 0 0 0
222 27 f 1 3 1
1 1 0 1 1 0 0 1 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 8 0 0 0
856 28 f 2 3 1
1 0 0 0 1 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 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 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 1 0 0
677 25 m 7 1.5 1
1 1 0 1 1 0 0 0 0 1 0 0 0
362 35 f .5 0 1
1 0 0 1 0 0 0 0 0 0 0 0 0 0
677 41 f 6 2 7
7 7 0 7 7 0 0 0 0 8 0 0 0
677 24 m 5 1 5
1 5 0 5 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 1 0 0 0 1 0 0 0 0
850 26 f 6 1 2 6
6 0 0 0 2 2 2 6 6 6 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
859 40 f 7 1 5
1 5 0 5 1 0 0 1 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 8 8 0 0
677 38 f 3 3 1
1 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 1 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
Appendix 1 • Raw Data and DATA Steps for Base SAS Procedures

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 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
362 27 f 1 1 1
1 0 1 0 1 4 0 4 1 0 4 4 0
967 32 f 3 2 7
7 0 0 0 7 0 0 1 0 0 1 0 0
362 29 f 1 0 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 0 0
546 32 m 5 .5 8
8 0 0 8 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 1 0
362 28 f 1 1 1
1 0 0 0 1 1 0 4 0 0 0 4 0
851 32 f .5 2 4
5 0 0 0 4 0 0 0 0 0 0 2 0
967 43 f 2 2 1
1 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 6
467 23 f 5 3 1
0 2 1 2 1 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
859 26 f 9.5 1.5 2
2 2 2 2 2 0 0 2 2 0 0 0 0
222 28 f 3 0 2
2 0 0 2 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
859 31 f 10 1 0 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
677 31 f .5 .5 6
7 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 2 0 0 0 0
781 24 f 3 3 6
1 6 6 6 1 6 0 0 0 1 0 1 1
856 27 f 1.5 1 6
2 6 6 6 2 5 0 2 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 0
382 30 m 1 2 4
7 0 0 0 7 0 4 7 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 2 0 0 0
849 29 m 10 1 5
7 5 5 5 7 5 5 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 6 7 7 7 7 7 6
222 32 f 10 5 1
1 1 0 1 1 0 0 1 1 1 0 1 0
677 32 f 1 0 1
1 0 1 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
677 30 m 4 6 1
7 0 0 0 0 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
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 5 0 0 0 0 0 0 0
783 40 f 1.5 3 1
1 0 0 0 1 4 0 0 1 1 0 0 0
851 30 f 1 1 6
6 0 0 0 6 0 0 6 0 6 0 0
851 40 f 1 1 5
5 0 0 0 5 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
467 37 f 4 8 1
1 0 0 0 1 0 3 0 3 1 1 0 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 2 0 0 0 0 0
859 23 f 8 3 3
3 2 0 2 3 0 0 1 0 3 0 0
967 31 f .5 0 1
1 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 2 0 2 0 0 0
848 33 f 2 2 7
7 0 0 0 7 0 7 7 0 0 7 0
688 35 f 5 8 3
2 2 2 2 2 0 0 3 3 3 3 3 0 0
Appendix 1 • Raw Data and DATA Steps for Base SAS Procedures

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 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 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 2 0 0 0
222 30 f 8 4 1
1 1 1 1 1 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 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 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 8 0 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 2 0 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
467 24 f 2 4 1
0 0 1 0 1 0 0 0 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 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 2 2 2 0 0 0
850 22 f 2 1 1
1 0 0 0 1 1 0 5 0 0 1 0 0
493 36 f 1 0 5
0 0 0 0 7 0 0 0 7 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 0 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 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 8 0 0 0 0
783 29 f 3 3 1
1 0 0 0 4 0 0 4 1 0 1 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
677 28 f 1 4 1
1 1 1 1 1 0 0 0 1 0 1 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 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 1
848 27 m 7 2 5
7 5 0 5 4 5 0 0 7 4 4 0 4
677 25 f 1 2 8
8 0 0 0 5 0 0 8 0 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
833 32 m 1 2 1
1 0 0 0 1 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
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 .................................................. 439

Third-Party Software Licenses ................................................... 440
1. Unicode Data Files and Software ........................................... 440
2. Chinese/Japanese Word Break Dictionary Data (cjdict.txt) ......... 441
3. Lao Word Break Dictionary Data (laodict.txt) ......................... 444
4. Time Zone Database ............................................................ 445

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2014 International Business Machines Corporation and others
All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS 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 THIS 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 this Software without prior written authorization of the copyright holder.
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

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.


# 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-------------------
#
# /*
# * Copyright (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-tsa4@uiuc.edu http://casper.beckman.uiuc.edu/~c-tsa4
#
# -------------------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,
oficers 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/
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 [RFC378] 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.
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 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

A

ALL option
  PROC JAVAINFO statement 218
ALTERNATE_HANDLING= option
  PROC SORT statement 333
APPEND procedure 3
  syntax 6
APPEND statement
  DATASETS procedure 47
appending data sets
  APPEND procedure versus APPEND statement 48
  block I/O method 4, 49
  compressed data sets 5, 49
  restricting rows 4
SET statement versus APPEND procedure 4
  system failures 6, 50
  variables with different attributes 5, 49
  with different variables 5
appending rows 3, 48
appending SAS data sets 7
appending tables 47
ASC
  ASCENDING option 345
ASCENDING option
  KEY statement (SORT) 345
ASCII collating sequence 322
ASCII option
  PROC SORT statement 331

B

BASE= argument
  APPEND procedure 7
APPEND statement (DATASETS) 47
BLANKLINE=
  PROC PRINT statement 243
block I/O method
  appening data sets 4, 49
BY groups
  complex transposition 369
  maintaining order of observations in 356
  retaining first observation of 359
  transposing 385
  transpositions with 374
BY statement
  ID statement (PRINT) with 253
  PRINT procedure 252
  SORT procedure 344
  TRANSPOSE procedure 374
BY variable
  printing unsorted data (PRINT) 253
BY variables
  single BY variable 355

C

CAS tables
  copying 54
CASE_FIRST= option
  PROC SORT statement 334
CASFMTLIB
  PROC FORMAT statement 104
catalog entries
  routing log or output to entries 307
catalogs
  format catalogs 98
  character strings
    formats for 133
  character values
    formats for 158
  character variables
    sorting orders for 322
CLASSPATHS option
  PROC JAVAINFO statement 218
CLONE option
  COPY procedure 38
  COPY statement (DATASETS) 16, 54
CNTLIN= option
  PROC FORMAT statement 104
CNTLOUT= option
  PROC FORMAT statement 105, 132
collating sequence 331
  alternate 332
ASCII 322
  based on National Use Differences 332
Danish 331
default 322
Finnish 331
Norwegian 331
specifying 332
specifying for character variables 323
Swedish 331
collating sequence options 332
collating sequences
creating 323
COLLATION= option
PROC SORT statement 334
column headings
customizing text in 263
page layout 240
column width 241
compressed data sets
appending 5, 49
concatenating data sets 8
CONTENTS procedure 13, 15
syntax 15
task table 15
versus CONTENTS statement (DATASETS) 52
CONTENTS statement
DATASETS procedure 50, 0
CONTENTS statement option
describing a table 25
using the DIRECTORY and DETAILS options 31
using the DIRECTORY option 29
CONTENTS= option
PROC PRINT statement 243
Copy
data set to a CAS table 41
COPY procedure 35
syntax 37
versus COPY statement (DATASETS) 35, 57
COPY statement
DATASETS procedure 53
TRANSPOSE procedure 375
copying CAS tables 54
COPY statement versus COPY procedure 57
copying data libraries
etire data library 36, 57
copying files 37
COPY statement versus COPY procedure 35
selected files 36
copying tables
excluding tables 59
selected tables 58, 60

D
DANISH option
PROC SORT statement 331
DATA argument
PROC DELETE 70
data encryption 182
data libraries
copying CAS tables 54
copying entire library 36, 57
copying files 37
deleting tables 58
printing directories of 13, 0
saving tables from deletion 60
data options
PROC IMPORT statement 199
data sets
appending compressed data sets 5, 49
concatenating 8
creating formats from 163
printing all data sets in library 291
sorting 320
DATA= argument
PROC EXPORT statement 75
DATA= option
APPEND procedure 7
APPEND statement (DATASETS) 48
CONTENTS statement (DATASETS) 16, 50
PROC PRINT statement 243
PROC SORT statement 338
PROC TRANSPOSE statement 372
DATAFILE= argument
PROC IMPORT statement 196
DATAROW statement
IMPORT procedure 199
DATASETS procedure 45
concepts 44
directory listings, as output 61
directory listings, to log 61
ending 44
execution of statements 44
ODS and 64
output tables 20, 65
procedure output 61
results 61
syntax 44
task tables 45, 0
DATATABLE= argument
PROC IMPORT statement 198
DATATYPE= option
PICTURE statement (FORMAT) 115
DATECOPY option
PROC SORT statement 338
DBENCODING statement
EXPORT procedure 77
IMPORT procedure 200
DBMS
  SORT procedure with 346
DBMS Identifiers Supported in SAS 76
DBMS= argument
  PROC EXPORT statement 76
  PROC IMPORT statement 198
DBMS= option
  PROC EXPORT statement 76
DD= option
  PROC DATASETS statement 46
DDNAME= option
  PROC DATASETS statement 46
DECSEP= option
  PICTURE statement (FORMAT) 115
DEFAULT= option
  FORMAT procedure 109, 115, 134
DEFINE option
  PROC OPTIONS statement 221
DELETE procedure
  concepts 69
  syntax 69
DELETE statement
  DATASETS procedure 58
deleting
  CAS tables 69
deleting CAS tables 70
delimited files
  exporting 78
  importing 200
  sharing across hosts 194
Delimited files
  TXT file example 87
DELIMITER statement
  EXPORT procedure 78
  IMPORT procedure 200
DELIMITER= option
  PROC TRANSPOSE statement 372
DESC
  DESCENDING option 346
DESCENDING option
  BY statement (PRINT) 253
  BY statement (SORT) 344
  BY statement (TRANSPOSE) 374
KEY statement (SORT) 346
DETAILS option
  CONTENTS statement (DATASETS) 19, 50
  PROC DATASETS statement 46
DIG3SEP= option
  PICTURE statement (FORMAT) 115
digit selectors 115
directives 115, 121
DIRECTORY option
  CONTENTS statement (DATASETS) 19, 51
disk space
  for SORT procedure 328
displaying system option information 225
DUPOUT= option
  PROC SORT statement 338

E
encoded passwords 313, 316
  encoding methods 315, 317
  in SAS programs 313
  saving to programs 316
encoding
  versus encryption 314
  encoding methods 315, 317
  encoding values 333
encryption
  versus encoding 314
EQUALS option
  PROC SORT statement 338
example data set 151
Excel
  importing spreadsheet from workbook 193
  importing subset of rows from 193
EXCLUDE statement
  DATASETS procedure 59
  FORMAT procedure 107
EXPORT procedure
  DBMS specifications 74
delimited files 73
  exporting delimited files 78, 79
  exporting to a CSV file 84
external data file
  copying 35, 37
FILE= option
  PROC PRINTTO statement 300
files
  copying 35, 37
FILL= option
  PICTURE statement (FORMAT) 115
Finnish collating sequence 331
FINnish option
  PROC SORT statement 331
FMTLEN option
CONTENTS statement (DATASETS) 19, 51

FMTLIB option
  PROC FORMAT statement 105, 132
FMTLIB statement
  EXPORT procedure 78
  IMPORT procedure 200

FORCE option
  APPEND procedure 7
  APPEND statement (DATASETS) 48
  PROC DATASETS statement 46
  PROC SORT statement 339

FORMAT procedure
  associating informats and formats with variables 97
  concepts 97
  excluding entries from processing 107
  format catalogs 98

FORMAT procedure
  associating informats and formats with variables 97
  concepts 97
  excluding entries from processing 107
  format catalogs 98
  overview 96
  printing informats and formats 100
  procedure output 144
  ranges 139
  results 141
  selecting entries for processing 132
  storing informats and formats 98
  syntax 102
  task tables 102, 103, 108, 113, 133
  values 139
  format-name formats 138
f
See also picture formats
  associating with variables 97
  creating drill-down tables 171
  creating from data sets 163
  creating in non-English languages 169
  for character values 133, 158
  format-name formats 138
  missing 100
  Perl regular expression 111
  permanent 99
  picture-name formats 132
  printing 100
  retrieving permanent formats 166
  temporary 99
  formatted values
    large dollar amounts 154
FUZZ= option
  PROC FORMAT procedure 109, 116, 134

GETNAMES statement
  IMPORT procedure 201
GRANDTOTAL_LABEL= option

PROC PRINT statement 244
  GROUP= option
  PROC OPTIONS statement 222
GTOTAL_LABEL= option
  PROC PRINT statement 244
GUESSINGROWS statement
  IMPORT procedure 201

headers
  response headers 187
HEADERS statement
  HTTP procedure 181
HEADING= option
  PROC PRINT statement 244
HELP option
  PROC JAVAINFO statement 218
HEXVALUE option
  PROC OPTIONS statement 222
HOST option
  PROC OPTIONS statement 222
  host sort utility 349
HTML5 reports 260
HTTP procedure 175
  capturing response headers 187
  POST request through proxy 186
  simple POST request 184
  syntax 176
HTTPS protocol 182
  making PROC HTTP calls with 182
  Hypertext Transfer Protocol Secure (HTTPS) 182

ID statement
  BY statement (PRINT) with 254
  PRINT procedure 253
  TRANSPOSE procedure 376
IDLABEL statement
  TRANSPOSE procedure 377
IMPORT procedure
  data source statements 193
  datafile/datatable 195
  delimited files 193
  generated SAS data sets 198
  input data 0
  JMP files 193
  META= statement 193
  syntax 195
In-Database Processing
  PROC SORT statement 346
IN= option
  COPY procedure 38
  COPY statement (DATASETS) 56
INDD= option
  COPY procedure 38
  COPY statement (DATASETS) 56
informs 96
  associating with variables 97
missing 100
  permanent 99
  printing 100
  raw data values 108
storing 98
  temporary 99
INLIB= option
  COPY procedure 38
  COPY statement (DATASETS) 56
input data sets
  presorted 324
input files
  importing external data files 195
  integrity constraints
  SORT procedure 348
INVALUE statement
  FORMAT procedure 108

J
Java environment 217
  JAVAINFO procedure 217
  JAVAINFO Procedure
    syntax 217
  JREOPTIONS option
    PROC JAVAINFO statement 218
JUST option
  INVALUE statement (FORMAT) 110

K
KEY statement
  SORT procedure 344
KILL option
  PROC DATASETS statement 46

L
LABEL option
  PROC PRINT statement 244
LABEL= option
  PROC PRINTTO statement 299
  PROC TRANSPOSE statement 372
LANGUAGE option
  PICTURE statement (FORMAT) 116
LET option
  PROC TRANSPOSE statement 373
LEVEL= option
  STRENGTH= option 335
LIB option
  PROC DATASETS statement 46

libraries
  printing all data sets 291
LIBRARY= option
  PROC DATASETS statement 46
  PROC FORMAT statement 105
Linguistic Sorting Using
  ALTERNATE_HANDLING= 361
Linguistic Sorting Using
  ALTERNATE_HANDLING= and
  STRENGTH=4 Option 363
LISTGROUPS option
  PROC OPTIONS statement 222
LISTINSERTAPPEND option
  PROC OPTIONS statement 222
LISTRESTRICT option
  PROC OPTIONS statement 222
LOCALE= option
  PROC SORT statement 335
log
  locations for 297
  restoring previous file location 302
  routing to catalog entries 307
  routing to external files 303
LOG= option
  PROC PRINTTO statement 299
LOGNUMBERFORMAT option
LONG option
  PROC OPTIONS statement 223

M
MAX= option
  FORMAT procedure 110, 117, 135
MAXLABELN= option
  PROC FORMAT statement 106
MAXSELEN= option
  PROC FORMAT statement 106
MEMTYPE= option
  COPY procedure 40
  COPY statement (DATASETS) 56
  EXCLUDE statement (DATASETS) 59
  SELECT statement (DATASETS) 60
message characters 115
METHOD= option
  PROC PWENCODE statement 315
MIN= option
  FORMAT procedure 110, 117, 135
missing informats and formats 100
missing values
  example 160
MOVE option
  COPY procedure 41
  COPY statement (DATASETS) 57
MULTILABEL option
  PICTURE statement (FORMAT) 117
VALUE statement (FORMAT) 135
MULTIPLIER= option
PICTURE statement (FORMAT) 118

N
N option
PROC PRINT statement 245
NAME= option
PROC PRINTTO statement 300
PROC TRANSPOSE statement 373
NATIONAL option
PROC SORT statement 332
National Use Differences 332
NEW option
PROC PRINTTO statement 300
NEW= option
APPEND procedure 7
APPEND statement (DATASETS) 48
NOBYLINE option
BY statement (PRINT) with 253
NODS option
CONTENTS statement (DATASETS) 19, 51
NODUPKEY option
PROC SORT statement 339
NOEDIT option
PICTURE statement (FORMAT) 119
NOEQUALS= option
PROC SORT statement 340
NOEXPAND option
PROC OPTIONS statement 223
NOHOST option
PROC OPTIONS statement 223
NOLIST option
PROC DATASETS statement 46
NOLOGNUMBERFORMAT option
PROC OPTIONS statement 223
NOOBS option
PROC PRINT statement 246
NOPRINT option
CONTENTS statement (DATASETS) 19, 51
PROC DATASETS statement 46
NOREPLACE option
PROC FORMAT statement 106
NORWEGIAN option
PROC SORT statement 331
NOTHREADS= option
PROC SORT statement 340
NOTSORTED option
BY statement (PRINT) 253
BY statement (TRANSPOSE) 374
FORMAT procedure 110, 119, 136
NOUNIKEY
NOUNIQUEKEY option 340
NOUNIKEYS
NOUNIQUEKEY option 340
NOUNIQUEKEY option 340
NOUNIQUEKEY option 340
NOWARN option
APPEND procedure 7
APPEND statement (DATASETS) 48
PROC DATASETS statement 47
National Use Differences 332
National Use Differences 332
sorting orders for numeric variables
summing 277
numeric values
summing 274
NUMERIC_COLLATION= option
PROC SORT statement 335

O
OBS= option
PROC PRINT statement 246
observations
maintaining order of, in BY groups 356
page layout 239
retaining first observation of each BY group 359
transposing variables into 367
ODS (Output Delivery System)
DATASETS procedure and 64
ODS table names
DATASETS procedure 64
OPTION= option
PROC OPTIONS statement 223
OPTIONS procedure
display settings for a group of options 227
overview 219
results 230
syntax 220
ORDER= option
CONTENTS statement (DATASETS) 19, 51
OS option
PROC JAVAINFO statement 218
OUT= argument
APPEND procedure 7
APPEND statement (DATASETS) 47
COPY procedure 37
COPY statement (DATASETS) 54
PROC IMPORT statement 197
OUT= option
CONTENTS statement (DATASETS) 19, 51
PROC PWENCODE statement 315
PROC SORT statement 340
PROC TRANSPOSE statement 373
OUTDD= argument
COPY procedure 37
COPY statement (DATASETS) 54
OUTFILE= option
PROC EXPORT statement 75
OUTLIB= argument
COPY procedure 37
COPY statement (DATASETS) 54
output files
exporting SAS data sets 74
OUTTABLE= option
PROC EXPORT statement 76
OVERRIDE= option
COPY procedure 41
COPY statement (DATASETS) 57
OVERWRITE option
PROC SORT statement 340

P
page ejects 255
page layout 239
column headings 240
column width 241
customizing 286
observations 239
with many variables 282
page numbering 302
PAGE option
PROC FORMAT statement 106
PAGEBY statement
PRINT procedure 255
partitioned data sets
threaded sorting 321
passwords
encoding 313, 316
encoding methods 317
paste buffer
saving encoded passwords to 316
PDF reports 263
performance tuning
for SORT procedure 328
permanent informats and formats 99
accessing 99
retrieving 166
picture formats 113
building 125
creating 152
digit selectors 115
directives 115
filling 156
message characters 115
PICTURE statement

P

INDEX

455

INDEX

455

FORMAT procedure 113
MULTIPLIER option 154
picture-name formats 132
PORTABLE option
OPTIONS procedure 223
PostScript files 282
PREFIX= option
PICTURE statement (FORMAT) 120
PROC TRANSPOSE statement 373
presorted input data sets 324
PRESORTED option
PROC SORT statement 341
PRINT procedure 260, 263
error processing 258
HTML5 reports 260
limit sums 256
overview 236
page ejection 255
page layout 239, 282, 286
PDF reports 263
PostScript files 282
procedure output 238
results 238
RTF reports 267
selecting variables 257
style elements 247
syntax 241
task tables 241
total numeric variables 255
XML files 274
PRINT= option
PROC PRINTTO statement 300
printing
all data sets in library 291
CAS tables 258
grouping rows 267
page ejects 255
page layout 239, 282, 286
restoring previous file location 302
selecting variables for 257, 260
table contents 13
template for printing numbers 113
PRINTTO procedure 297
overview 297
restoring log and output file locations 302
syntax 297
PROC CONTENTS statement 15
PROC DATASETS statement
DATASETS procedure 45
PROC DELETE
DELETE procedure 70
PROC EXPORT statement 74
PROC FORMAT statement 103
PROC HTTP calls 182
PROC HTTP statement
HTTP procedure 176
PROC IMPORT statement 196
PROC JAVAINFO statement 217
PROC OPTIONS statement 220
PROC PRINT statement 242
PROC PRINTTO statement 298
PROC PWENCODE statement
  PWENCODE procedure 314
PROC SORT statement
  SORT procedure 330
PROC TRANPOSE statement 372
procedure output
  locations for 297
  page numbering 302
  routing to catalog entries 307
  routing to external files 303
  procedures
    raw data for examples 398
PRODUCT_STATUS procedure 311
  syntax 311
proxy servers 186
punctuating numbers 223
PUTNAMES statement
  EXPORT procedure 78
PWENCODE procedure 313
  concepts 313
  encoding methods 317
  encoding passwords 316
  encoding versus encryption 314
  saving encoded passwords to paste
    buffer 316
  syntax 314

R
ranges
  FORMAT procedure 139
raw data
  informats for 108
  procedures examples 398
REALMEMSIZE system option 327
RECALL command
  IMPORT procedure, Delimited files 194
REGEXP
  INVALIDE value (FORMAT) 111
REGEXPE option
  INVALIDE value (FORMAT) 111
REPLACE option
  PROC EXPORT statement 77
  PROC IMPORT statement 198
reports 236, 263
  customized 237
  grouping rows 267
  limiting sums in 282
PDF 263
RTF 267
  selecting variables for 257
response headers 187
restoring previous output file location 302
restoring previous SAS log file location 302
RESTRICT option
  PROC OPTIONS statement 223
REVERSE option
  PROC SORT statement 332
ROUND option
  PICTURE statement (FORMAT) 120
  PROC PRINT statement 246
rows
  appending 3, 48
  grouping for reports 267
RTF reports 267

S
SAS data sets
  appending 7
SAS programs
  encoded passwords in 313
SAVE statement
  DATASETS procedure 60
SELECT statement
  DATASETS procedure 60
  FORMAT procedure 132
SET statement
  appending data 4
SHORT option
  CONTENTS statement (DATASETS)
    20, 51
  PROC OPTIONS statement 223, 224
single BY variables
  creating views 355
SIZE=
  SORTSIZE= option 341
sort order
  for character variables 322
  for numeric variables 322
SORT procedure
  character variable sorting orders 322
  collating sequence 322, 331
  collating sequence option 332
  concepts 321
  creating collating sequences 323
  DBMS data source 346
disk space for 328
encoding values 333
host sort utility 349
integrity constraints 348
LINGUISTIC collation 333
Linguistic Sorting Using
   ALTERNATE_HANDLING=
Linguistic Sorting Using
   ALTERNATE_HANDLING= and
   STRENGTH=4 Option
maintaining order of observations in BY
groups
numeric variable sorting orders
output
output data set
performance tuning for
presorted input data sets
results
retaining first observation of each BY
group
sorting by values of multiple variables
sorting data sets
sorting in descending order
stored sort information
syntax
task tables
threaded sorting
translation tables
sort utility
sorting, threaded
SORTSEQ= option
PROC SORT statement
SORTSIZE= option
PROC SORT statement
SPLIT= option
PROC PRINT statement
spreadsheets
importing from Excel workbook
importing subset of rows from
SSL
see TLS
statistical analysis
transposing data for
stored sort information
STRENGTH= option
PROC SORT statement
style elements
PRINT procedure
STYLE= option
ID statement (PRINT)
PROC PRINT statement
SUM statement (PRINT)
VAR statement (PRINT)
SUFFIX= option
PROC TRANSPOSE statement
SUM statement
BY statement (PRINT) with
PRINT procedure
SUMBY statement
PRINT procedure
SUMLABEL option
PROC PRINT statement
Swedish collating sequence
SWEDISH option
PROC SORT statement
system failures
system options
display setting for single option
display settings for a group
displaying a list
displaying information about
displaying restricted options
list of current settings
OPTIONS procedure
short form listing
T
tables
appending
content descriptions
contents of
deleting
excluding from copying
saving from deletion
selecting for copying
tagsort option
PROC SORT statement
templates
for printing numbers
temporary informats and formats
threaded sorting
threads
THREADS option
SORT procedure
tLS
translation tables
TRANSPOSE procedure
attributes of transposed variables
complex transposition
copying variables without transposing
duplicate ID values
formatted ID values
labeling transposed variables
listing variables to transpose
naming transposed variables
output data set
output data set variables
results
simple transposition
syntax
transposing BY groups
transposing data for statistical analysis 389
transposition types 368
transpositions with BY groups 374
variable names, from numeric values 377
transposed variables 368
attributes of 379
labeling 378, 383
naming 379, 381, 387
trust stores 182

U
UCA
LINGUISTIC option 333
UNIOUT
UNIQUEOUT= option 343
PROC SORT statement 343
UNIT= option
PROC PRINTTO statement 302
UPCASE option
INVALUE statement (FORMAT) 111

V
VALUE option
PROC OPTIONS statement 224
VALUE statement

W
Web service
invoking 0

X
XML files 274