



SAS Publishing

The SYLK Procedure (Experimental)



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2002. *The SYLK Procedure (Experimental)*. Cary, NC: SAS Institute Inc.

The SYLK Procedure (Experimental)

Copyright © 2002 by SAS Institute Inc., Cary, NC, USA

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of the software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, July 2002

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at www.sas.com/pubs or call 1-800-727-3228.

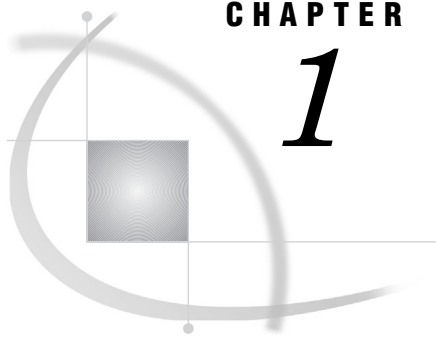
SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.® indicates USA registration.

IBM® and DB2® are registered trademarks or trademarks of International Business Machines Corporation. ORACLE® is a registered trademark of Oracle Corporation. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Contents

Chapter 1	△	The SYLK Procedure (Experimental)	1
Overview:		SYLK Procedure	1
Syntax:		SYLK Procedure	2
PROC SYLK-Specific Programming Statements			6
Predefined SYLK Functions			14
Concepts:		SYLK Procedure	17
Examples:		SYLK Procedure	30



CHAPTER

1

The SYLK Procedure (Experimental)

<i>Overview: SYLK Procedure</i>	1
<i>Syntax: SYLK Procedure</i>	2
<i>PROC SYLK Statement</i>	2
<i>READ Statement</i>	3
<i>BY Statement</i>	5
<i>PROC SYLK-Specific Programming Statements</i>	6
<i>ARRAY statement</i>	6
<i>ATTRIB Statement</i>	7
<i>DO Statement, Iterative</i>	8
<i>DROP Statement</i>	8
<i>FUNCTION Statement</i>	8
<i>IF-THEN/ELSE Statement</i>	9
<i>KEEP Statement</i>	9
<i>OUTARGS Statement</i>	10
<i>OUTPUT Statement</i>	10
<i>PUT Statement</i>	11
<i>RETAIN Statement</i>	12
<i>RETAINBY Statement</i>	12
<i>SUBROUTINE Statement</i>	13
<i>Predefined SYLK Functions</i>	14
<i>ACELL Functions</i>	14
<i>RCELL Functions</i>	14
<i>SLKBLOCK Function</i>	15
<i>Miscellaneous Predefined PROC SYLK Functions</i>	16
<i>Concepts: SYLK Procedure</i>	17
<i>Programming in PROC SYLK</i>	17
<i>Automatic Variables in PROC SYLK</i>	17
<i>Using PROC SYLK Directly as a Spreadsheet</i>	18
<i>Using PROC SYLK to Read a SYLK File</i>	19
<i>Microsoft Excel Formula and Format Translation</i>	21
<i>Examples: SYLK Procedure</i>	30
<i>Example 1: Reading a SYLK file</i>	30
<i>Example 2: BY-Group Processing with PROC SYLK</i>	34
<i>Example 3: Subsetting the Data and Adding a Column</i>	38
<i>Example 4: Using PROC SYLK with PROC REPORT</i>	40

Overview: SYLK Procedure

CAUTION:

PROC SYLK is an experimental procedure that is available in Version 9. Do not use this procedure in production jobs. △

The SYLK procedure reads an external SYLK-formatted spreadsheet into SAS. A SYLK file is an ASCII file that has a standard format for storing the data, formulas, and formatting from a spreadsheet application. Many spreadsheet applications can export spreadsheets in SYLK format.

While SAS has always had the ability to import data from various external spreadsheets, PROC SYLK takes this capability one step further by capturing the functionality and formatting of the external spreadsheets as well. This capability enables you to use an external spreadsheet application (such as Microsoft Excel) to initially design and lay out a complicated table or report. The spreadsheet can then be saved in SYLK format and automatically associated with SAS data sets that would be prohibitively large to use in most GUI-based spreadsheets.

The SYLK procedure can also be used directly as a batch spreadsheet. When using PROC SYLK as a batch spreadsheet, you use programming statements to manipulate data, perform calculations, generate summaries, and format the output. Programming in PROC SYLK is analogous to using formulas and formats in a spreadsheet application.

With PROC SYLK, whether you import an external spreadsheet or build a spreadsheet directly, you can select a subset of observations from the input data set, define the format of a table by using the Output Delivery System, operate on its row and column values, and create new columns and rows. Access to individual table values is also available when needed.

Syntax: SYLK Procedure

Tip: Supports the Output Delivery System. See *SAS Output Delivery System User's Guide* for details.

```
PROC SYLK <option(s)>;
  READ SYLK=sylk-filename <option(s)>;
  BY <DESCENDING> variable(s);
  programming-statements
```

PROC SYLK Statement

```
PROC SYLK <option(s)>;
```

Options

```
DATA= SAS-data-set
```

specifies a SAS data set that contains the input data. By default, all variables in the input data set correspond to columns in the spreadsheet. The PROC SYLK programming statements are executed once for each observation in the input data set. If DATA= is not specified, then the program is executed only once.

END=variable

creates and names a temporary variable that contains an end-of-file indicator. The variable, which is initialized to 0, is set to 1 when PROC SYLK reads the last observation of the input data set. This variable is not added to the output data set.

LABEL=package-label

specifies a descriptive label for the package to which compiled functions and subroutines are stored.

Interaction: The LABEL= option has no effect unless the OUTCAT= option is also specified.

LIBRARY=subroutine-catalog | (subroutine-catalog-list)

specifies a previously compiled catalog or catalogs to be searched for functions and subroutines that were created in a previous PROC SYLK step. Subroutine catalogs are created by the OUTLIB= option and stored as members of a SAS catalog with type CMPSUB.

You can specify the name of a subroutine catalog as *libref.catalog.entry* or *libref.catalog*. If the entry is not specified, then all the CMPSUB entries in *libref.catalog* are used.

You can specify a list of files in the LIBRARY= option, and you can specify a range of names with numeric suffixes; here's an example:

```
library=Sub1-Sub10
```

When you specify more than one file, except in the case of a single range of names, you must enclose the list in parentheses; here's an example:

```
library=(a b c)
```

Alias: INLIB=

OUT=SAS-data-set

specifies a SAS data set to which the results of executing the program are to be written. All variables that are used in the program and not specified in a DROP statement are written to the OUT= data set whenever an OUTPUT statement is executed.

OUTLIB=libref.catalog.package

specifies the name of a package to which compiled subroutines are to be stored at the end of the PROC SYLK step. You must specify the full three-level name.

READ Statement

Reads in a SYLK-formatted file and translates it to corresponding SAS code.

Interaction: When the READ statement is used, all other statements in the current PROC SYLK step are ignored.

Featured in: Example 1 on page 30

```
READ SYLK=sylk-filename <option(s)>;
```

Required Argument

SYLK=<'>*syLK-file*<'>

specifies which SYLK-formatted spreadsheet to import. *syLK-file* can be a fileref or a physical filename. If *syLK-file* is enclosed in single (') or double (") quotation marks, then PROC SYLK assumes it is a physical filename. Otherwise, PROC SYLK assumes it is a fileref. If no such fileref exists, then PROC SYLK looks for a physical file by that name (with a **.s1k** extension) in the current directory. If no file by that name is found, then PROC SYLK stops processing and writes an error message to the log.

Options

HEADERS=*n* | NOHEADERS

specifies whether or not the SYLK file contains column headings, and if so, specifies the number (*n*) of rows that contain column headings. If HEADERS=*n* is specified, then PROC SYLK concatenates the column heading text from the first *n* rows of each column into a label for the corresponding variable in the SAS data set. If NOHEADERS is specified but text headers are encountered in the SYLK file, then an error will result because numeric SAS variables must not contain character data.

NOHEADERS has the same effect as HEADERS=0.

Default: HEADERS=1

HTML=*HTML-filename*

specifies the name of a file that is specified for the FILE= option on the ODS HTML statement in the generated SAS program. A **.htm** extension is added to *HTML-filename*. If the HTML= option is not specified, then no ODS HTML statement is included in the generated SAS program. Do not enclose *HTML-filename* in quotation marks.

Note: Currently, you can specify only the name of the HTML file for the HTML= option; you cannot specify a path. To add a path specification, edit the ODS HTML statement in the generated SAS program. \triangle

Alias: HTMLFILE=

OUT=*SAS-data-set*

specifies the name for the output SAS data set that is produced when a SYLK-formatted file is translated. All static data cells are written to this data set. Any cell that contains a formula is initially set to missing in this data set, but its value is calculated when the PROC SYLK step executes.

Default: SASUSER.SYLK

Interaction: The TXTFILE= and OUT= options are mutually exclusive. If both are specified, then the OUT= option is ignored.

SASFILE=*file*

specifies the file that contains the SAS program that is generated when a SYLK-formatted file is translated. *file* can be a fileref or a physical filename. If *file* is a physical filename, then PROC SYLK automatically adds a **.sas** extension to the filename. If the SASFILE= option is not specified, then the filename **syLK.sas** is used. Unless another path is specified, PROC SYLK stores the file in the current directory.

TITLE=*'title'*

specifies a title that is used for the ODS output table that is created. *title* can be enclosed in either single (') or double (") quotation marks.

Default: If the TITLE= option is not specified, then a default title of “Imported SYLK File *sylk-file*” is used.

TEXTFILE=*file*

specifies an output text file to use instead of an output SAS data set when a SYLK-formatted file is translated. *file* can be either a fileref or a physical filename. If an output text file is specified, then extra DATA step code is included in the generated SAS program file in order to import this data to a SAS data set before the PROC SYLK step.

Interaction: The TEXTFILE= and OUT= options are mutually exclusive. If both are specified, then the OUT= option is ignored.

BY Statement

Establishes BY groups and sets up special grouping variables for the input data set.

Restriction: The input data set must be sorted by the variables that are specified in the BY statement.

Featured in: Example 2 on page 34

BY <DESCENDING> *variable(s)*;

Required Argument

variable(s)

specifies the variables to use to define groups within the input data set. The input data set must be sorted by the BY variables.

Option

DESCENDING

indicates that the input data set is sorted in descending order by the specified variable. DESCENDING means largest to smallest for numeric variables or reverse alphabetical order for character variables.

Details

Whenever a BY statement is used, a `_LAST_byvar` variable is automatically created that indicates the last iteration for that BY group. This variable enables you to do summations or other user-specific processing over each BY group. All `_LAST_byvar` variables are dropped from the output data set.

PROC SYLK-Specific Programming Statements

ARRAY statement

Associates a name with a list of variables, constants, or both.

```
ARRAY array-name <[dimension-size(s)]> <variables <(initial-values)>> | <constants>;
```

Required Argument

array-name

names the array.

Restriction: *array-name* must be a SAS name that is not the name of a SAS variable in the same PROC SYLK step.

Options

dimension-size(s)

specifies the number of elements in each dimension of the array. For arrays with more than one dimension, the dimension sizes are separated by commas. An array can have up to six dimensions. *dimension-size(s)* can be enclosed in either square brackets ([]) or braces ({}).

Restriction: The *dimension-size(s)* must be specified if a list of elements (variables or constants) is not specified.

variables

specifies a list of variables.

initial-values

specifies initial values for the corresponding elements in the array. The initial values can be numbers or character strings. Character strings must be enclosed in quotation marks.

constants

specifies constant values for the corresponding values in the array. Array elements that have constants assigned to them in the ARRAY statement cannot later have different values assigned to them.

Details

The ARRAY statement is similar to, but not identical to, the ARRAY statement in the DATA step. The ARRAY statement is used to associate a name with a list of variables and/or constants. The array name can then be used with subscripts in the program to refer to the items in the list.

The ARRAY statement in PROC SYLK does not support all the features of the ARRAY statement in the DATA step. Implicit indexing variables cannot be used; all array references must have explicit subscript expressions. Only simple array dimensions are allowed; lower bound specifications are not supported.

The ARRAY statement in PROC SYLK does support some features that are not supported in the ARRAY statement in the DATA step. PROC SYLK's ARRAY statement allows both variables and constants to be used as array elements. Both the dimension specification and the list of elements are optional, but at least one of these must be given. When the list of elements is not given, or when fewer elements than the size of the array are listed, array variables are created by suffixing element numbers to the array name to complete the element list.

Examples

The following ARRAY statement creates array A and assigns the constant values 1 through 10 to its ten elements:

```
array a[10] 1 2 3 4 5 6 7 8 9 10;
```

The following ARRAY statement creates array S and assigns variables S1 through S94 to its 94 elements:

```
array s[94] s1-s94;
```

The following ARRAY statement creates array OPTS, assigns variables OPT1 through OPT3 to its three elements, and initializes their values:

```
array opts[3] opt1-opt3 (1 1.0e-12 100);
```

ATTRIB Statement

Associates one or more attributes with one or more variables.

ATTRIB *variable(s) attribute-list(s);*

Required Arguments

variable(s)

names the variable(s) that you want to associate with the attributes.

attribute-list(s)

specifies one or more attributes to assign to the *variable(s)*. Specify one or more of these attributes in the ATTRIB statement:

FORMAT=*format*

associates a format with the *variable(s)*.

LABEL=*'label'*

associates a label with the *variable(s)*.

LENGTH=<\$>*length*

associates a length with the *variable(s)*. Use the dollar sign (\$) to indicate character variables.

Examples

Here are three examples of ATTRIB statements.

- Assigning multiple attributes to one variable:

```
attrib x1 format=date7. label='variable x1'
```

- Assigning the same set of attributes to multiple variables:

```
attrib x1 x2 x3 label= 'sample size' length=4;
```

- Assigning different attributes to different variables:

```
attrib x1 format=date7. label='transaction date'
      x2 length=5
      x3 label= 'quantity' format=4.
      x4 length=$2 format=$4.;
```

DO Statement, Iterative

Executes statements between DO and END repetitively based on the value of an index variable.

```
DO index-variable=specification-1 <, ... specification-n>;
  ... programming statements ...
END;
```

The iterative DO statement in PROC SYLK has the same characteristics as the iterative DO statement in the DATA step, except that *index-variable* must be a numeric variable.

DROP Statement

Drops the specified variables from the output data set.

```
DROP variable(s)
```

Required Argument

variable(s)

specifies which variables to drop from the output data set. Any variables that are specified in the BY statement are automatically included in the output data set, even if they are listed in the DROP statement.

FUNCTION Statement

Defines a function.

```
FUNCTION function-name <(argument-1<, ... argument-n>)> <LABEL='label'>;
  ... programming-statements ...
RETURN (expression);
ENDSUB;
```

Required Arguments

expression

is an expression that resolves to a value that the function returns.

function-name

names the function.

programming-statements

are one or more programming statements.

Optional Arguments

argument

defines an argument, or input, to the function.

LABEL=*label*

is a descriptive label for the function. *label* must be no more than 256 characters long and must be enclosed in either single (') or double (") quotation marks.

Details

The FUNCTION statement is used to create a callable function. Functions are equivalent to routines as used in many other programming languages. They are independent computational blocks that require zero or more arguments. All variables that are declared within a function block are local to that function. You can declare, compile, and save any number of functions.

IF-THEN/ELSE Statement

Executes a programming statement for observations that meet specific conditions.

IF *expression* **THEN** *statement*;

<**ELSE** *statement*; >

The IF-THEN/ELSE statement in PROC SYLK has the same characteristics as the IF-THEN/ELSE statement in the DATA step, except that *expression* cannot resolve to a character value. For example, the following statement is not valid in PROC SYLK:

```
if 'this' then do;
```

KEEP Statement

Keeps the specified variables in the output data set.

KEEP *variable(s)*

Required Argument

variable(s)

specifies which variables to write to the output data set. Any variables that are specified in the BY statement are automatically included in the output data set, even if they are not listed in the KEEP statement.

OUTARGS Statement

Specifies arguments whose values can be changed by a subroutine or function.

Valid in: SUBROUTINE and FUNCTION

OUTARGS *argument-1*<, ... *argument-n*>;

Arguments

argument

is one of the arguments that are specified in the function or subroutine definition.

Details

OUTARGS lists arguments to a subroutine or function whose values can be changed by that subroutine or function. If an argument to a subroutine or function is not listed in an OUTARGS statement, and if a statement subroutine or function attempts to change the value of that argument, then the value is not changed.

OUTPUT Statement

Writes an observation to the output data set.

OUTPUT <*output-option*>;

Option

output-option

can be either `_ROW_` or one or more slashes (/).

`_ROW_`

writes the entire observation to the output data set.

/

writes an observation with all missing values to the output data set. You can use any number of slashes in a single OUTPUT statement to indicate the number of such observations to write.

Default: `_ROW_`

PUT Statement

Writes lines to the procedure output file.

PUT *<specification(s)>* *<@|@@>*;

The PUT statement in PROC SYLK is typically used as a debugging aid for PROC SYLK programs. It has the same characteristics as the PUT statement in the DATA step, with these exceptions:

- The lines of the PUT statement in PROC SYLK are written to the procedure output file, which by default appears in the SAS Output window. You can direct the lines to the SAS log by using the following FILE statement:

```
file log;
```

- The PUT statement in PROC SYLK does not support line pointers, factored lists, iteration factors, overprinting, `_INFILE_`, the colon (:) format modifier, or the special character \$.
- The PUT statement in PROC SYLK does not support subscripted array names unless they are enclosed in parentheses. For example, the statement

```
put (a[3]);
```

prints the third element of the array A, but the statement

```
put a[3];
```

results in an error message.

- The PUT statement in PROC SYLK does not recognize the asterisk (*) subscript, but an array name can be used in a PUT statement without subscripts. For example, if A is the name of an array, then the statement

```
put a;
```

is acceptable, but the statement

```
put a*;
```

is invalid.

- You can use the equal sign (=) after any variable name to print the variable with each value labeled with the name of the variable. Here's an example:

```
put a=;
```

- The PUT statement in PROC SYLK supports expressions inside parentheses. For example, the statement

```
put (sqrt(x));
```

prints the square root of X.

- The PUT statement in PROC SYLK supports the print item `_PDV_` in order to print a formatted listing of all the variables in the program. The statement

```
put _pdv_;
```

prints a more readable listing of the variables than is printed by the statement

```
put _all_;
```

RETAIN Statement

Causes a variable to retain its value from one iteration of the program to the next.

Featured in: Example 2 on page 34

RETAIN *variable(s)* <*initial-value(s)*>;

Required Argument

variable(s)

specifies variables whose values are to be retained from one processed observation to the next.

Optional Argument

initial-value(s)

specifies the initial values of the *variable(s)*. The variables are reinitialized at each BY-group boundary unless they are specified in a RETAINBY statement. If you omit *initial-value(s)*, then the initial value(s) are set to missing.

Examples

```
retain a b c 0;
retain a 100 b 200 c 300;
retain x0-x10 34.567;
```

RETAINBY Statement

Causes a variable to retain its value from one BY group to the next.

Featured in: Example 2 on page 34

RETAINBY *by-variable*=(*variable(s)* *initial-value(s)*);

Required Arguments

by-variable

specifies the BY variable that defines BY groups across which the values of the *variable(s)* are to be retained.

variable(s)

specifies the variables whose values are to be retained from one processed observation to the next and across BY-group boundaries that are defined by values of *by-variable* (and any lower-order BY variables).

initial-value(s)

specifies the initial values of the *variable(s)*. The variables are reinitialized at BY-group boundaries that are defined by BY variables that are of a higher order than *by-variable*.

Example

In this example, the BY variables Company, Division, and Department are declared, with Company being the highest-order BY variable. The retained variables Div_Total and Div_Mean are reinitialized when the value of Company changes, but not when the value of Division or Department changes. The retained variables Comp_Total and Comp_Mean are not reinitialized because no BY variable is of a higher order than Company.

```
by company division department;
retainby company=(comp_total 0 comp_mean 0);
retainby division=(div_total 0 div_mean 0);
```

SUBROUTINE Statement

Defines a subroutine.

```
SUBROUTINE subroutine-name <(argument-1<, ... argument-n>)> <LABEL='label'>;
... programming-statements...
ENDSUB;
```

Required Arguments***subroutine-name***

names the subroutine.

programming-statements

are one or more programming statements.

Optional Arguments***argument***

defines an argument, or input, to the subroutine.

label

is a descriptive label for the subroutine. *label* must be no more than 256 characters long and must be enclosed in either single (') or double (") quotation marks.

Details

The SUBROUTINE statement is used to create callable subroutines. Subroutines are similar to functions except that they have no return value. Subroutines are called with the CALL statement.

Predefined SYLK Functions

ACELL Functions

Retrieve a value from a given variable and observation.

ACELLC (*column*, *row*);

ACELLN (*column*, *row*);

Functions

- The ACELLC function returns a character value.
- The ACELLN function returns a numeric value.

Arguments

column

names the variable from which the value is to be retrieved.

row

indicates the observation number from which the value is to be retrieved. *row* must be a positive integer.

RCELL Functions

Return a value from a variable for an observation that is relative to the current observation.

RCELLC (*column*, *offset*);

RCELLN (*column*, *offset*);

Functions

- RCELLC returns a character value.
- RCELLN returns a numeric value.

Arguments

column

names the variable from which to retrieve the value.

offset

when added to the number of the current observation, indicates the number of the observation from which to retrieve the value. *offset* must be a nonpositive integer. A negative value indicates a lower-numbered observation; a value of 0 indicates the current observation.

Note: Currently, the RCELL functions can retrieve values only from lower-numbered observations or the current observation. For this reason, *offset* must be a nonpositive integer. Δ

SLKBLOCK Function

Performs a given function over one or more ranges of cells.

SLKBLOCK(*function-name*, *block-parameter-1*<, ... , *block-parameter-n*>);

Arguments

function-name

is the name of the function to be applied. *function-name* can be the name of a SAS function, one of the miscellaneous predefined PROC SYLK functions, or a user-defined function. If the function is a user-defined function, it must be a numeric function that can take any number of numeric arguments. *function-name* must be enclosed in single (') or double (") quotation marks.

block-parameter

specifies a value or range of values. You can include any number of *block-parameter* specifications, separated by commas. A *block-parameter* can be

- a literal numeric value
- a numeric expression
- a variable name, which returns the value of that variable for the current observation
- the string '**of variable**' (including the single quotation marks), which returns all values of *variable* up to and including the current observation
- the string '**abs(first-column, first-observation, last-column, last-observation)**' (including the single quotation marks), which returns all values in the given rectangular block, where the observation specifications are absolute
- the string '**rel(first-column, first-observation, last-column, last-observation)**' (including the single quotation marks), which returns all values in the given rectangular block, where the observation specifications are relative to the current observation.

Details

For the '**abs()**' and '**rel()**' block parameters, the *first-column* and *last-column* specifications indicate the leftmost and rightmost columns, respectively, to be processed. *first-column* and *last-column* can be column names or column numbers.

For the '**abs()**' block parameter, the *first-observation* and *last-observation* specifications indicate the top and bottom observations to be processed. *first-observation* and *last-observation* must be integers that represent observation numbers.

For the `'rel()'` block parameter, the *first-observation* and *last-observation* specifications indicate the top and bottom observations, relative to the current observation, to be processed. *first-observation* and *last-observation* must be nonpositive integers because the `'rel()'` block parameter can reference only lower-numbered observations.

Examples

This example sums the values in a rectangular block from the first column through the fourth column and from the second observation through the sixth observation:

```
total = slkblock('sum', 'abs(1, 2, 4, 6)');
```

This example sums the values in a rectangular block from the SALARY1 column through the SALARY4 column and from the second observation through the sixth observation:

```
total = slkblock('sum', 'abs(salary1, 2, salary4, 6)');
```

This example sums the values in a rectangular block from the SALARY1 column through the SALARY4 column and from six observations up from the current observation to two observations up from the current observation:

```
total = slkblock('sum', 'rel(salary1, -6, salary4, -2)');
```

This example sums the values of the SALARY column up to and including the current observation:

```
total = slkblock('sum', 'of salary');
```

This example sums the values of the SALARY column from two observations up from the current observation through the current observation:

```
total = slkblock('sum', 'rel(salary, -2, salary, 0)');
```

This example applies the MEAN function across the last six observations of the SALARY and COMP columns:

```
average = slkblock('mean', 'rel(salary, -6, salary, 0)',  
                  'rel(comp, -6, comp, 0)');
```

This example applies the MEAN function to all values of SALARY, the current value of COMP, two literal numeric values, and a numeric expression:

```
average = slkblock('mean', 'of salary', comp, 27, 28, 29*2);
```

Miscellaneous Predefined PROC SYLK Functions

Numerous predefined functions are provided in the SASHELP.SLKWXL catalog to support Microsoft Excel functions that do not have a corresponding SAS function. PROC SYLK automatically searches this catalog for available functions. The functions in this catalog are generally named *excel-function-name_slk*. See “Microsoft Excel Formula and Format Translation” on page 21 to see which Microsoft Excel functions are supported in this way.

Concepts: SYLK Procedure

Programming in PROC SYLK

Programming statements are used in the main body of the PROC SYLK step or in subroutine and function blocks to manipulate data and calculate values in the same way that a spreadsheet application does. A PROC SYLK program executes once for each observation in the input data set. The results are written to the output data set when the OUTPUT statement is called.

When using PROC SYLK directly as a spreadsheet program, you write the PROC SYLK program yourself. When reading a SYLK-formatted file with the READ SYLK statement, PROC SYLK generates the programming statements for you, using the information in the SYLK file.

In addition to the PROC SYLK-specific programming statements that are described in “PROC SYLK-Specific Programming Statements” on page 6, the following programming statements are available. They behave in the same way that they do in the DATA step, as documented in *SAS Language Reference: Dictionary*:

ABORT	LINK
assignment	OTHERWISE
CALL	RETURN
DELETE	SELECT
DO	STOP
DO UNTIL	WHEN
DO WHILE	sum
GOTO	

In addition to the programming statements, you can use SAS functions, PROC SYLK functions, and user-defined functions to perform calculations and operations.

Automatic Variables in PROC SYLK

PROC SYLK creates the following automatic variables any time that it processes an input data set:

`_N_`

contains the number of the current observation in the input data set.

`_ROW_`

contains the number of the current observation in the output data set. Generally, `_ROW_` equals `_N_`. However, if rows are being skipped for output or extra rows are being inserted as the input data set is processed, then `_N_` and `_ROW_` become unequal.

`_LAST_by-variable`

indicates whether or not PROC SYLK is currently processing the last observation for a particular BY variable group. See “BY Statement” on page 5 for more details.

`_NOBS_`

indicates the total number of observations in the input data set.

Using PROC SYLK Directly as a Spreadsheet

The following example shows how PROC SYLK can be used directly as a spreadsheet. In the example, a data set is created, and then PROC SYLK performs simple summations over rows and columns and creates an output data set.

```

data report;
  input compdiv $ date:date7. salary travel insure advrtise;
  format date date9.;
  datalines;
A 31JAN2002 95000 10500 2000 6500
B 31JAN2002 668000 112000 5600 90000
C 31JAN2002 105000 6800 9000 18500
A 28FEB2002 91000 8200 1900 12000
B 28FEB2002 602000 99000 5500 86000
C 28FEB2002 96000 6000 8500 16000
;

proc sylk data=report out=rep_out end=last;
  totalcost= salary + travel + (insure * 2) + advrtise;
  output;
  if last then do;
    output /;
    compdiv = '';
    date = .;
    salary = COLSUM(salary, ., .);
    travel = COLSUM(travel, ., .);
    insure = COLSUM(insure, ., .);
    advrtise = COLSUM(advrtise, ., .);
    totalcost = COLSUM(totalcost, ., .);
    output;
  end;
run;

ods html file= "report.htm";
proc print data= rep_out;
run;
ods html close;

```

In this example, the columns of the table correspond to the variables in the input data set and the rows correspond to the observations. By default, all newly introduced variables in the program, such as TotalCost, are added to the columns of the output table as well. Each row (or observation) is written to the output data set with the additional column TotalCost by using the OUTPUT statement. The TotalCost column is calculated with a programming line that performs a mathematical operation on the preceding four columns. Column values can also be calculated by using SAS functions or user-defined functions. Finally, after the last row is written to the output data set, column summations are performed and written. The following HTML output is produced by this program:

The SAS System

Obs	compdiv	date	salary	travel	insure	advrtise	totalcost
1	A	31JAN2002	95000	10500	2000	6500	116000
2	B	31JAN2002	668000	112000	5600	90000	881200
3	C	31JAN2002	105000	6800	9000	18500	148300
4	A	28FEB2002	91000	8200	1900	12000	115000
5	B	28FEB2002	602000	99000	5500	86000	798000
6	C	28FEB2002	96000	6000	8500	16000	135000
7
8	.	.	1657000	242500	32500	229000	2193500

Using PROC SYLK to Read a SYLK File

To import a SYLK-formatted spreadsheet file into SAS, you use the SYLK procedure with the READ SYLK statement. PROC SYLK generates a SAS program that contains

- a PROC TEMPLATE step that defines the formatting for the output
- a PROC SYLK step that includes all the programming statements that are required to reproduce the functionality of the original spreadsheet
- ODS HTML statements and a DATA _NULL_ step that generate the output.

Note: PROC SYLK ignores any Microsoft Excel macros that are included in the SYLK file. Δ

For this example, the SYLK file (ex1.slk) that is created from the Microsoft Excel spreadsheet in the following display is read into SAS. A SAS program file and an output data set (sasuser.ex1) are created.

	A	B	C	D	E	F	G	H
1	compdiv	date	salary	travel	insure	advertise		
2	A	31-Jan-02	95000	10500	2000	6500	116000	
3	B	31-Jan-02	668000	112000	5600	90000	881200	
4	C	31-Jan-02	105000	6800	9000	18500	148300	
5	A	28-Feb-02	91000	8200	1900	12000	115000	
6	B	28-Feb-02	602000	99000	5500	86000	798000	
7	C	28-Feb-02	96000	6000	8500	16000	135000	
8								
9			1657000	242500	32500	229000	2193500	
10								

```
proc sylk;
  read sylk=ex1.slk
      out=sasuser.ex1
      html=ex1;
run;
```

By default, PROC SYLK creates a SAS program file with the name **sylik.sas**. The SAS program file contains the following program:

```

/*- SAS file transferred from .SYLK file -*/

options noovp;

ods path sasuser.tmpl(update) sashelp.tmplmst(read);

/*- create Table Template -*/
proc template;
  define table table.sylk;
    translate _val_=. into "";
    column x1 x2 x3 x4 x5 x6 x7 _ROW_;

    define x1; label= ''; end;
    define x2; format= ddmmyy.; label= 'date'; end;
    define x3; label= ''; end;
    define x4; label= ''; end;
    define x5; label= ''; end;
    define x6; label= ''; end;
    define _ROW_; label='row'; end;

    header tablettitle;
    define tablettitle;
      text 'Imported SYLK File - ex1.slk';
    end;

  end;
run;

proc sylk data=sasuser.ex1 out=sasuser.ex1_out end=last;
  if _row_ ge 1 and _row_ le 6 then x7 = x3+x4+(2*x5)+x6;
  if _row_ eq 8 then do;
    x3 = SLKBLOCK("SUM", "REL(x3,-7,x3,-2)");
    x4 = SLKBLOCK("SUM", "REL(x4,-7,x4,-2)");
    x5 = SLKBLOCK("SUM", "REL(x5,-7,x5,-2)");
    x6 = SLKBLOCK("SUM", "REL(x6,-7,x6,-2)");
    x7 = SLKBLOCK("SUM", "REL(x7,-7,x7,-2)");
  output;
  end;
  else output;
run;

/*- print final table -*/
ods html file='ex1.htm';
data _null_; set sasuser.ex1_out;
  file print ods=(template='table.sylk');
  put _ods_;
run;
ods html close;

```

When you run the generated SAS program, the following HTML output is produced:

The SAS System						
Imported SYLK File - ex1.slk						
compdiv	date	salary	travel	insure	advertise	x7
A	31-01-02	95000	10500	2000	6500	116000
B	31-01-02	668000	112000	5600	90000	881200
C	31-01-02	105000	6800	9000	18500	148300
A	28-02-02	91000	8200	1900	12000	115000
B	28-02-02	602000	99000	5500	86000	798000
C	28-02-02	96000	6000	8500	16000	135000
		1657000	242500	32500	229000	2193500

Notice that the PROC SYLK step in this SAS program file is similar to the SAS program in “Using PROC SYLK Directly as a Spreadsheet” on page 18. The HTML output from the two programs is similar as well. When PROC SYLK imports a SYLK file, it always generates a PROC TEMPLATE step in order to retain the formatting of the original spreadsheet. See *SAS Output Delivery System User’s Guide* for details about the TEMPLATE procedure. If the HTML= option is specified on the READ statement, then PROC SYLK generates an ODS HTML statement to produce the output, and generates a DATA _NULL_ step in order to print the output data set. A DATA _NULL_ step is used because PROC PRINT does not support ODS templates.

You can edit the SAS program file in order to customize the output, change the path of the HTML file, or use a different ODS destination.

Microsoft Excel Formula and Format Translation

The following tables describe how PROC SYLK translates Microsoft Excel formulas and formats into SAS functions and formats. If no SAS function is indicated for a given Excel function, then that Excel function is not implemented in PROC SYLK; attempting to read a SYLK file that contains that Excel function will result in an error. You can avoid this problem by writing a user-defined function that has the same name and arguments as the Excel function.

Table 1.1 Math and Trigonometry Functions

Excel Function	SAS Function
ABS	ABS
ACOS	ARCOS
ACOSH	ARCOSH
ASIN	ARSIN
ASINH	ARSINH
ATAN	ATAN
ATAN2	ATAN2

Excel Function	SAS Function
ATANH	ARTANH
CEILING(x,sg)	CEIL(x/sg)* sg
COMBIN(N, k)	COMB(N, k)
COS	COS
COSH	COSH
COUNTIF	
DEGREES(x)	$x*180/CONSTANT('PI')$
EVEN	EVEN_SLK
EXP	EXP
FACT	FACT
FLOOR	FLOOR_SLK
GCD	GCD
INT	INT
LCM	
LN	LOG
LOG(N, a)	LOG(N)/LOG(a)
LOG10	LOG10
MDETERM	
MINVERSE	
MMULT	
MOD	MOD
MROUND(a,b)	ROUND($a,ABS(b)$)
MULTINOMIAL(a, b, c)	GAMMA($a+b+c$)/ (GAMMA($a+1$)*GAMMA($b+1$)*GAMMA($c+1$))
ODD	ODD_SLK
PI()	CONSTANT('PI')
POWER(a, b)	$a**b$
PRODUCT	PRODUCT_SLK
QUOTIENT(n, d)	INT(n,d)
RADIANS(x)	$x*CONSTANT('PI')/180$
RAND()	RANUNI(_seed_)
RANDBETWEEN(a,b)	INT((RANUNI(_seed_)+ a)*($b-a$))
ROMAN(x,y)	PUT($x, ROMAN.$)
ROUND(d,n)	ROUND($d, 10**(-n)$)
ROUNDDOWN(d,n)	INT($d*10**n$)/ $10**n$
ROUNDUP(d,n)	SIGN(d)*CEIL(ABS(d)* $10**n$)/ $10**n$
SERIESSUM	

Excel Function	SAS Function
SIGN	SIGN
SIN	SIN
SINH	SINH
SQRT	SQRT
SQRTPI(x)	SQRT($x*\text{CONSTANT}('PI')$)
SUBTOTAL	
SUM($arguments$)	SLKBLOCK('SUM', $arguments$)
SUMIF	
SUMPRODUCT	
SUMSQ	
SUMX2MY2(x,y)	USS(x) – USS(y)
SUMX2PY2(x,y)	USS(x) + USS(y)
SUMXMY2(x,y)	USS($x - y$)
TAN	TAN
TANH	TANH
TRUNC(x)	INT(x)
TRUNC(x, y)	INT($x*10^{**y}$)/ 10^{**y}

Table 1.2 Statistical Functions

Excel Function	SAS Function
AVEDEV	AVDEV_SLK
AVERAGE	MEAN
AVERAGEA	
BETADIST($x, p1, p2, a, b$)	PROBBETA($(x-a)/(b-a), p1, p2$)
BETAINV(p, a, b, a, b)	BETAINV($(x-a)/(b-a), p1, p2$)
BINOMDIST($N, nt, p, cm=1$)	PDF('BINOMIAL', N, p, nt)
BINOMDIST($N, nt, p, cm=0$)	CDF('BINOMIAL', N, p, nt)
CHIDIST()	1 – PROBCHI()
CHIINV($p0, \dots$)	CINV($1-p0, \dots$)
CHITEST	
CONFIDENCE	
CORREL	
COUNT	N
COUNTA	
COVAR	
CRITBINOM	

Excel Function	SAS Function
DEVSQ	DEVSQ_SLK
FINV($p0, p1, p2$)	FINV($1-p0, p1, p2$)
FISHER	
FISHERINV	
FORECAST	
FREQUENCY	
FTEST	
GAMMADIST($x, a, b, cm=1$)	CDF('GAMMA', x, a, b)
GAMMADIST($x, a, b, cm=0$)	PDF('GAMMA', x, a, b)
GAMMAINV($x, a, b=1$)	GAMINV(x, a)
GAMMALN	LGAMMA
GEOMEAN	
GROWTH	
HARMEAN	
HYPGEOMDIST($x0, x1, x2, x3$)	PROBHYPR($x3, x2, x1, x0$)– PROBHYPR($x3, x2, x1, x0-1$)
INTERCEPT	
KURT	KURTOSIS
LARGE($x0, x1, x2, \dots n$)	LARGEST(ROUND($n+0.5$), $x0, x1, x2, \dots$)
LINEST	
LOGEST	
LOGINV	QUANTILE('LOGNORMAL' ...)
LOGNORMDIST($p0, p1, p2$)	CDF('LOGNORMAL', $p0, p1, p2$)
MAX	MAX
MAXA	
MEDIAN	MEDIAN
MIN	MIN
MINA	
MODE	
NEGBINOMDIST(nf, ns, p)	PDF('NEGBINOMIAL', nf, p, ns)
NORMDIST($x, \mu, \sigma, cm=1$)	CDF('NORMAL', x, μ, σ)
NORMDIST($x, \mu, \sigma, cm=0$)	PDF('NORMAL', x, μ, σ)
NORMINV	
NORMSDIST	PROBNORM
NORMSINV	PROBIT
PEARSON	
PERCENTILE($x0, x1, \dots n$)	PCTL($n*100, x0, x2, \dots$)
PERCENTRANK	

Excel Function	SAS Function
PERMUT(N, k)	PERM(INT(N), INT(k))
POISSON($x, mean, cm=1$)	CDF('POISSON', $x, mean$)
POISSON($x, mean, cm=0$)	PDF('POISSON', $x, mean$)
PROB	
QUARTILE	
RANK	
RSQ	
SKEW	SKEWNESS
SLOPE(x, y)	
SMALL(x_0, x_1, x_2, \dots, n)	SMALLEST(ROUND($n-0.5$), x_0, x_1, x_2, \dots)
STANDARDIZE(x, μ, σ)	$(x-\mu)/\sigma$
STDEV	STD
STDEVA	
STDEVPA	SQRT($(n-1)/n$)*STD where n is the number of samples
STDEVX	
TDIST($x, fdg, cm \neq 2$)	1-PROBT(x, fdg)
TDIST($x, fdg, cm=2$)	1-PROBT(x, fdg)*2
TINV(p, fdg)	TINV($1-p/2, fdg$)
TREND	
TRIMMEAN	
TTEST	
VAR	VAR
VARA	
VARP	VARP_SLK
VARPA	
WEIBULL($x, \alpha, \beta, cum=1$)	CDF('WEIBULL', x, α, β)
WEIBULL($x, \alpha, \beta, cum=0$)	PDF('WEIBULL', x, α, β)
ZTEST	

Table 1.3 Text Functions

Excel Function	SAS Function
CHAR	BYTE
CLEAN	
CODE("CHAR")	RANK('CHAR')
CONCATENATE(T1,T2 ...)	T1 T2 ...

Excel Function	SAS Function
DOLLAR	
EXACT(text1,text2)	text1 == text2
FIND	
FIXED	
LEFT('abc')='a'	
LEN	LENGTH
LOWER	LOWCASE
MID	
PROPER	
REPLACE	
REPT(text, n)	
RIGHT('abc')='c'	
SEARCH()	
SUBSTITUTE()	
T	
TEXT	
TRIM	
UPPER	UPCASE
VALUE(text)	

Table 1.4 Financial Functions

Excel Function	SAS Function
ACCRINT	ACCRINT_SLK
ACCRINTM	ACCRINTM_SLK
AMORDEGRC	AMORDEGRC_SLK
AMORLINC	AMORLINC_SLK
COUPDAYBS	COUPDAYBS_SLK
COUPDAYS	COUPDAYS_SLK
COUPDAYSNC	COUPDAYSNC_SLK
COUPNCD	COUPNCD_SLK
COUPNUM	COUPNUM_SLK
COUPPCD	COUPPCD_SLK
CUMIPMT	-CUMNIPMT
CUMPRINC	-CUMPRINC
DB	DB_SLK
DDB(c, s, l, p, r)	DEPDB(p, c-s, l, r)

Excel Function	SAS Function
DISC	DISC_SLK
DOLLARDE	DOLLARDE_SLK
DOLLARFR	DOLLARFR_SLK
DURATION	DURATION_SLK
EFFECT	EFFECT_SLK
FV	
IPMT	-IPMT
MDURATION	MDURATION_SLK
NPV	
ODDFPRICE	ODDFPRICE_SLK
ODDFYIELD	ODDFYIELD_SLK
ODDLPRICE	ODDLPRICE_SLK
ODDLYIELD	ODDLYIELD_SLK
PMT	-PMT
PPMT	-PPMT
PRICE	PRICE_SLK
PRICEDISC	PRICEDISC_SLK
PRICEMAT	PRICEMAT_SLK
PV	
RATE	
RECEIVED	RECEIVED_SLK
SLN(<i>inico</i> st, <i>salvage</i> , <i>life</i>)	
SYD(<i>cost</i> , <i>salvage</i> , <i>life</i> , <i>per</i>)	
TBILLEQ	TBILLEQ_SLK
TBILLPRICE	TBILLPRICE_SLK
TBILLYIELD	TBILLYIELD_SLK
VDB(<i>c</i> , <i>s</i> , <i>l</i> , <i>p0</i> , <i>p1</i> , <i>r</i> , 1)	DACCDB(<i>p1</i> , <i>c-s</i> , <i>l</i> , <i>r</i>)-DACCDB(<i>p0</i> , <i>c-s</i> , <i>l</i> , <i>r</i>)
VDB(<i>c</i> , <i>s</i> , <i>l</i> , <i>p0</i> , <i>p1</i> , <i>r</i> , 0)	DACCDBSL(<i>p1</i> , <i>c-s</i> , <i>l</i> , <i>r</i>)-DACCDBSL(<i>p0</i> , <i>c-s</i> , <i>l</i> , <i>r</i>)
XIRR	
XNPV	
YIELD	YIELD_SLK
YIELDDISC	YIELDDISC_SLK
YIELDMAT	YIELDMAT_SLK

Table 1.5 Format Conversions

Excel Format String	SAS Format
@	\$w.
General	BEST.
0	w.d
0.00	w.d
#,##0	COMMAw.d
#,##0.00	COMMAw.d
#,##0_);(,##0)	NEGPARENw.d
#,##0_);[Red](,##0)	NEGPARENw.d
#,##0.00_);(,##0.00)	NEGPARENw.d
#,##0.00_);[Red](,##0.00)	NEGPARENw.d
\$\$,##0_);(\$,##0)	DOLLARw.d
\$\$,##0_);[Red](\$,##0)	DOLLARw.d
(\$,##0.00_);(\$,##0.00)	DOLLARw.d
(\$,##0.00_);[Red](\$,##0.00)	DOLLARw.d
\$*#,##0);_(\$*(#,##0);_(\$*"_"_);_(@_)	DOLLARw.d
*#,##0);_*(#,##0);_(*"_"_);_(@_)	NEGPARENw.d
\$*#,##0.00);_(\$*(#,##0.00);_(\$*"-"??_);_(@_)	DOLLARw.d
*#,##0.00);_*(#,##0.00);_(*"-"??_);_(@_)	NEGPARENw.d
0%	PERCENTw.d
0.00%	PERCENTw.d
0.00E+00	Ew.d
##0.0E+0	Ew.d
m/d/yy	MMDDYYw.
d-mmm-yy	MMDDYYw.
d-mmm	DATEw.
mmm-yy	MONYYw.
h:mm AM/PM	TIMEw.
h:mm:ss AM/PM	TIMEw.
h:mm	TIMEw.
hh:mm	TIMEw.
h:mm:ss	TIMEw.
hh:mm:ss	TIMEw.
m/d/yy h:mm	DATETIMEw.
ddmmyy	DATEw.
ddmmyyyy:hh:mm:ss	DATETIMEw.
dd	DATEw.

Excel Format String	SAS Format
dd/mm/yy	DDMMYYw.
dddd	DATEw.
mm/dd/yy	MMDDYYw.
mm:ss	MMSSw.
mm yy	MONYYw.
mm yyyy	MONYYw.
mm:yy	MONYYw.
mm:yyyy	MONYYw.
mm-yy	MONYYw.
mm-yyyy	MONYYw.
mmyy	MONYYw.
mmyyyy	MONYYw.
mm.yy	MONYYw.
mm.yyyy	MONYYw.
mm/yy	MONYYw.
mm/yyyy	MONYYw.
mmmm	MONYYw.
m	MONYYw.
mmyy	MONYYw.
mmyyyy	MONYYw.
dddd, mmmm dd, yyyy	MONYYw.
dddd, dd mmmm yyyy	MONYYw.
mmmm dd, yyyy	MMDDYYw.
dd mmmm yyyy	MONYYw.
yy	YYMMDDw.
yyyy	YYMMDDw.
yy mm	YYMMDDw.
yyyy mm	YYMMDDw.
yy:mm	YYMMDDw.
yyyy:mm	YYMMDDw.
yy-mm	YYMMDDw.
yyyy-mm	YYMMDDw.
yymm	YYMMDDw.
yyyymm	YYMMDDw.
yy.mm	YYMMDDw.
yyyy.mm	YYMMDDw.
yy/mm	YYMMDDw.

Program

Set the SAS system options.

```
options nodate pageno=1 linesize=64 pagesize=60;
```

Read the external SYLK file. The OUT= option specifies the SAS data set that is to be created. The TITLE= option specifies a title that is used in the ODS output table. The HTML= option specifies the filename for the HTML output file.

```
proc sylk;
  read sylk=inventory.slk
      out=sasuser.inv_q1
      html=inv_q1
      title='First Quarter Inventory';
run;
```

Print the SASUSER.INV_Q1 data set.

```
proc print data=sasuser.inv_q1;
run;
```

Output

Notice that the first two observations contain all missing values, because the spreadsheet file has two blank rows between the column headings and the data. Notice also that the X8 variable contains missing values, as do the X2 through X7 variables for observation 17. These values are calculated by the PROC SYLK step in the generated SAS program, and the calculated values are included in the output data set.

The SAS System								1
Obs	x1	x2	x3	x4	x5	x6	x7	x8
1
2
3	Arlington	AlSaw	993000	59.99	75	30	75	.
4	Bainbridge	AlSaw	207688	89.99	211	220	295	.
5	Arlington	AlSaw	308979	999.99	345	320	532	.
6	Bainbridge	AlSaw	612386	125.99	123	150	495	.
7	Arlington	Abrasive	132543	19.99	34	60	183	.
8	Bainbridge	Abrasive	694348	15.99	756	700	734	.
9	Arlington	AlSaw	705407	10.99	869	800	1556	.
10	Bainbridge	Abrasive	355605	14.99	567	600	1469	.
11	Arlington	Abrasive	375805	5.99	345	330	897	.
12	Bainbridge	Abrasive	385905	6.99	4230	4000	4345	.
13	Arlington	Abrasive	400312	7.99	5400	5000	9230	.
14	Bainbridge	Abrasive	365705	8.99	2123	1800	7200	.
15	Arlington	Abrasive	395000	9.99	4590	4100	6223	.
16	Bainbridge	AlSaw	500750	5.99	234	200	4790	.
17	Total:

Generated Program

The following program is generated by PROC SYLK and stored as external file INVENTORY.SAS. For details about the ODS statements and PROC TEMPLATE, see *SAS Output Delivery System User's Guide*.

Set the SAS system option NOOVP. The NOOVP option ensures that errors that are identified in the SAS log are indicated by dashes in the line below the error.

```
/*- SAS file transferred from .SYLK file -*/

options noovp;
```

Specify where to search for template definitions.

```
ods path sasuser.tmpl(update) sashelp.tmplmst(read);
```

Define the ODS template.

```
/*- create Table Template -*/
proc template;
  define table table.sylk;
    translate _val_ into "";
    column x1 x2 x3 x4 x5 x6 x7 x8 _ROW_;

    define x1; label= 'Store'; end;
    define x2; label= 'Product'; end;
    define x3; label= 'Cat. No.'; end;
    define x4; label= 'Unit Price'; end;
    define x5; label= 'January'; end;
    define x6; label= 'February'; end;
    define x7; label= 'March'; end;
    define x8; label= 'Season'; end;
    define _ROW_; label='row'; end;

    header tabletitle;
    define tabletitle;
      text 'First Quarter Inventory';
    end;

  end;
run;
```

Read the input data set. The DATA= option indicates which SAS data set to use for processing. The OUT= option names the output SAS data set. The END= option creates a variable, Last, that contains the value 1 when the last observation in the input data set is read.

```
proc sylk data=sasuser.inv_q1 out=sasuser.inv_q1_out end=last;
```

Calculate seasonal totals for each row. For each row of data, the Season column (X8) is calculated as the sum of the values of the January, February, and March columns (X5, X6, and X7).

```
if _row_ ge 3 and _row_ le 16 then x8 = SLKBLOCK("SUM",of x5-x7);
```

Calculate totals for each month and for the entire quarter. For the last row, the January, February, March, and Season (X5, X6, X7, and X8) columns are summed.

```
if _row_ eq 17 then do;
  x5 = SLKBLOCK("SUM", "REL(x5,-14,x5,-1)");
  x6 = SLKBLOCK("SUM", "REL(x6,-14,x6,-1)");
  x7 = SLKBLOCK("SUM", "REL(x7,-14,x7,-1)");
  x8 = SLKBLOCK("SUM", "REL(x8,-14,x8,-1)");
  output;
end;
else output;
run;
```

Set the HTML output filename.

```
/*- print final table -*/
ods html file='inv_q1.htm';
```

Generate the output HTML file. The FILE PRINT statement specifies the ODS template to use in formatting the output. The PUT _ODS_ statement physically writes each line to the output file.

```
data _null_;
  set sasuser.inv_q1_out;
  file print ods=(template='table.sylk');
  put _ods_;
run;
```

Close the HTML destination.

```
ods html close;
```

Output from Generated Program

The SAS System							
First Quarter Inventory							
Store	Product	Cat. No.	Unit Price	January	February	March	Season
Arlington	A1Saw	993000	59.99	75	30	75	180
Bainbridge	A1Saw	207688	89.99	211	220	295	726
Arlington	A1Saw	308979	999.99	345	320	532	1197
Bainbridge	A1Saw	612386	125.99	123	150	495	768
Arlington	Abrasive	132543	19.99	34	60	183	277
Bainbridge	Abrasive	694348	15.99	756	700	734	2190
Arlington	A1Saw	705407	10.99	869	800	1556	3225
Bainbridge	Abrasive	355605	14.99	567	600	1469	2636
Arlington	Abrasive	375805	5.99	345	330	897	1572
Bainbridge	Abrasive	385905	6.99	4230	4000	4345	12575
Arlington	Abrasive	400312	7.99	5400	5000	9230	19630
Bainbridge	Abrasive	365705	8.99	2123	1800	7200	11123
Arlington	Abrasive	395000	9.99	4590	4100	6223	14913
Bainbridge	A1Saw	500750	5.99	234	200	4790	5224
Total:				19902	18310	38024	76236

Example 2: BY-Group Processing with PROC SYLK

Procedure features:

- BY statement
 - RETAIN statement
 - RETAINBY statement
 - Programming statements
-

The code in this example replaces the PROC SYLK step in “Generated Program” on page 32. The resulting program

- generates subtotals for each combination of store and product
- generates subtotals for each store for all products
- generates grand totals for all stores.

Program

Copy the SASUSER.INV_Q1 data set, eliminating the Totals row and any blank rows.

The last row of the original spreadsheet is for calculating totals and contains the value **Totals:** in the first column. Because this value would be seen as a store name by the PROC SYLK step, it must be eliminated. Similarly, the first two rows of the original spreadsheet are blank; the missing value in the first column would be seen as a store name by the PROC SYLK step and must be eliminated. The LENGTH statement increases the length of the X1 variable so that it can contain the value **GRAND TOTAL:**.

```
data sasuser.inv_q1_sort;
  length x1 $ 12;
  set sasuser.inv_q1;
  if x1 ne "Total:" and x1 ne "";
run;
```

Sort the SASUSER.INV_Q1_SORT data set. The SORT procedure sorts the input data set by values of the Store and Product columns.

```
proc sort data=sasuser.inv_q1_sort;
  by x1 x2;
run;
```

Read the sorted data set. The DATA= option indicates the input data set that PROC SYLK is to use in processing. The OUT= option names the output SAS data set. The END= option creates a variable, Last, that contains the value 1 when the last observation in the input data set is read.

```
proc sylk data=sasuser.inv_q1_sort out=sasuser.inv_q1_out end=last;
```

Establish BY groups. The BY statement sets up BY groups by store and by product within each store.

```
by x1 x2;
```

Create retained variables. The values of Jan_prod, Feb_prod, Mar_prod, and Season_prod are initialized to 0, retained, and reset to 0 at the beginning of each new BY group. These variables store running totals for each item for each month.

```
retain jan_prod feb_prod mar_prod season_prod 0;
```

Create variables to be retained within BY groups. The Jan_store, Feb_store, Mar_store, and Season_store values are initialized to 0 and retained across changes in the Product column (X2); they are reset to 0 when the value of Store (X1) changes. These variables store running totals for each store for each month. The Jan_gtotal, Feb_gtotal, Mar_gtotal, and Season_gtotal values are initialized to 0 and retained across changes in the Store column (X1). These variables store running totals for the entire data set.

```
retainby x2=(jan_store feb_store mar_store season_store 0);
retainby x1=(jan_gtotal feb_gtotal mar_gtotal season_gtotal 0);
```

Calculate seasonal totals for each item.

```
x8 = SLKBLOCK("SUM", of x5-x7);
```

Calculate running totals for each item. The OUTPUT statement writes the current values of X1 through X8 to the output data set.

```
jan_prod += x5;
feb_prod += x6;
mar_prod += x7;
season_prod += x8;
output;
```

Generate subtotals for each product in each store. The automatic variable `_LAST_X2` is set to 1 when the current row is the last one in the BY group for that product. After the assignment statements update the values of X1 through X8, the OUTPUT statement writes them to the output data set.

```
if _last_x2 then do;
  x1 = UPCASE(x1);
  x2 = UPCASE(x2);
  x3 = .;
  x4 = .;
  x5 = jan_prod;
  x6 = feb_prod;
  x7 = mar_prod;
  x8 = season_prod;
output;
```

Calculate running totals for each store.

```
jan_store += x5;
feb_store += x6;
mar_store += x7;
season_store += x8;
```

Generate subtotals for each store. The value of `_LAST_X1` is set to 1 when the input row is the last one for that store. After the assignment statements update the values of X1 through X8, the OUTPUT statement writes them to the output data set.

```
if _last_x1 then do;
  x1 = UPCASE(x1);
  x2 = "";
  x3 = .;
  x4 = .;
  x5 = jan_store;
  x6 = feb_store;
  x7 = mar_store;
  x8 = season_store;
output;
```

Calculate grand totals over all stores. The OUTPUT statement writes a blank line (an observation with all missing values) to the output data set.

```

        jan_gtotal += x5;
        feb_gtotal += x6;
        mar_gtotal += x7;
        season_gtotal += x8;
    end;
    output /;
end;
```

Generate grand totals. When the last input observation is read, the value of LAST is set to 1. After the assignment statements update the values of X1 through X8, the OUTPUT statement writes them to the output data set.

```

    if last then do;
        x1 = "GRAND TOTAL:";
        x2 = "";
        x3 = .;
        x4 = .;
        x5 = jan_gtotal;
        x6 = feb_gtotal;
        x7 = mar_gtotal;
        x8 = season_gtotal;
        output;
    end;
run;
```

Output

First Quarter Inventory							
Store	Product	Cat. No.	Unit Price	January	February	March	Season
Arlington	A1Saw	993000	59.99	75	30	75	180
Arlington	A1Saw	308979	999.99	345	320	532	1197
Arlington	A1Saw	705407	10.99	869	800	1556	3225
ARLINGTON	A1SAW			1289	1150	2163	4602
Arlington	Abrasive	132543	19.99	34	60	183	277
Arlington	Abrasive	375805	5.99	345	330	897	1572
Arlington	Abrasive	400312	7.99	5400	5000	9230	19630
Arlington	Abrasive	395000	9.99	4590	4100	6223	14913
ARLINGTON	ABRASIVE			10369	9490	16533	36392
ARLINGTON				11658	10640	18696	40994
Bainbridge	A1Saw	207688	89.99	211	220	295	726
Bainbridge	A1Saw	612386	125.99	123	150	495	768
Bainbridge	A1Saw	500750	5.99	234	200	4790	5224
BAINBRIDGE	A1SAW			568	570	5580	6718
Bainbridge	Abrasive	694348	15.99	756	700	734	2190
Bainbridge	Abrasive	355605	14.99	567	600	1469	2636
Bainbridge	Abrasive	385905	6.99	4230	4000	4345	12575
Bainbridge	Abrasive	365705	8.99	2123	1800	7200	11123
BAINBRIDGE	ABRASIVE			7676	7100	13748	28524
BAINBRIDGE				8244	7670	19328	35242
GRAND TOTAL:				19902	18310	38024	76236

Example 3: Subsetting the Data and Adding a Column

Procedure features:

- WHERE statement
 - Subsetting IF statement
 - Assignment statement
-

This example replaces the PROC SYLK step in “Generated Program” on page 32. The resulting program

- processes only the rows where the unit price is less than 100
- calculates a new Sales column based on the unit price and the quantity sold.

Program

Read the input data set. The DATA= option indicates which SAS data set to use for processing. The OUT= option names the output SAS data set. The END= option creates a variable, Last, that contains the value 1 when the last observation in the input data set is read.

```
proc sylk data=sasuser.inv_q1 out=sasuser.inv_q1_out end=last;
```

Subset the input data set. The subsetting IF statement eliminates the blank rows and the row that is used for totals in the original spreadsheet. The WHERE statement limits the input rows to those in which the unit price is less than 100.

```
  if x1 ne "Total:" and x1 ne "";  
  where x4 < 100.0;
```

Specify attributes for the new Sales column. The ATTRIB statement specifies a format and label for the column.

```
  attrib sales format=dollar15.2 label="Sales";
```

Calculate seasonal totals for each row.

```
  x8=sum(of x5-x7);
```

Calculate Sales. Sales is calculated as the unit price (x4) multiplied by the seasonal total (x8).

```
  sales = x4 * x8;
```

Write the output row.

```
  output;  
run;
```

Output

The SAS System							
First Quarter Inventory							
Store	Product	Cat. No.	Unit Price	January	February	March	Season
Arlington	A1Saw	993000	59.99	75	30	75	180
Bainbridge	A1Saw	207688	89.99	211	220	295	726
Arlington	Abrasive	132543	19.99	34	60	183	277
Bainbridge	Abrasive	694348	15.99	756	700	734	2190
Arlington	A1Saw	705407	10.99	869	800	1556	3225
Bainbridge	Abrasive	355605	14.99	567	600	1469	2636
Arlington	Abrasive	375805	5.99	345	330	897	1572
Bainbridge	Abrasive	385905	6.99	4230	4000	4345	12575
Arlington	Abrasive	400312	7.99	5400	5000	9230	19630
Bainbridge	Abrasive	365705	8.99	2123	1800	7200	11123
Arlington	Abrasive	395000	9.99	4590	4100	6223	14913
Bainbridge	A1Saw	500750	5.99	234	200	4790	5224

Example 4: Using PROC SYLK with PROC REPORT

Procedure features:

This example

- modifies the PROC SYLK step in “Generated Program” on page 32 in order to create a simple SAS data set
- sorts the output data set
- replaces the DATA _NULL_ step in “Generated Program” on page 32 with a PROC REPORT step in order to generate the output report.

Program

Generate the output data set. This PROC SYLK step creates a SAS data set that contains only the input data and a seasonal total column.

```
proc sylk data=sasuser.inv_q1 out=sasuser.inv_q1_out end=last;
  if x1 ne "Total:" and x1 ne "";
  attrib sales format=dollar15.2 label="Sales";
  x8=sum(of x5-x7);
  sales = x4 * x8;
  output;
run;
```

Specify the HTML filename.

```
ods html file="ExcelEx3.htm";
```

Generate the report. For more information about PROC REPORT, see **Base SAS Procedures Guide**.

```
proc report data=sasuser.inv_q1_out nowd;
  define x1 / order;
  define x2 / order;
  define x3 / display;
  define x4 / display;
  define x5 / analysis sum;
  define x6 / analysis sum;
  define x7 / analysis sum;
  define x8 / analysis sum;
  define sales / analysis sum;
  break after x1 / summarize
      skip
      style(SUMMARY)={ foreground=green font_weight=bold };
run;
```

Close the HTML destination.

```
ods html close;
```

Output

The SAS System								
Store	Product	Cat. No.	Unit Price	January	February	March	Season	Sales
Arlington	A1Saw	993000	59.99	75	30	75	180	\$10,798.20
		308979	999.99	345	320	532	1197	\$1,196,988.03
		705407	10.99	869	800	1556	3225	\$35,442.75
	Abrasive	132543	19.99	34	60	183	277	\$5,537.23
		375805	5.99	345	330	897	1572	\$9,416.28
		400312	7.99	5400	5000	9230	19630	\$156,843.70
		395000	9.99	4590	4100	6223	14913	\$148,980.87
Arlington				11658	10640	18696	40994	\$1,564,007.06
Bainbridge	A1Saw	207688	89.99	211	220	295	726	\$65,332.74
		612386	125.99	123	150	495	768	\$96,760.32
		500750	5.99	234	200	4790	5224	\$31,291.76
	Abrasive	694348	15.99	756	700	734	2190	\$35,018.10
		355605	14.99	567	600	1469	2636	\$39,513.64
		385905	6.99	4230	4000	4345	12575	\$87,899.25
		365705	8.99	2123	1800	7200	11123	\$99,995.77
Bainbridge				8244	7670	19328	35242	\$455,811.58

Your Turn

If you have comments or suggestions about *The SYLK Procedure (Experimental)*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
email: yourturn@sas.com

Send suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
email: suggest@sas.com