CHAPTER 14 Functions and CALL Routines

SAS Functions under OS/2 **245**
SAS CALL Routines under OS/2 **245**
BYTE **245**
COLLATE **249**
DMYTECHC **250**
DMYTECWD **251**
DMYTERVC **252**
MCIPISLP **253**
MCIPISTR **254**
MODULExy **255**
PEEK **257**
RANK **258**
SLEEP **258**
TRANSLATE **259**
WAKEUP **260**

SAS Functions under OS/2

SAS functions and Call routines return a value from a computation or system operation. Most functions and Call routines use arguments that are supplied by the user as input.

Most SAS functions are completely described in the SAS functions and CALL routines portion of *SAS Language Reference: Dictionary*. The functions that are described here have syntax or behavior that is specific to the OS/2 operating environment.

SAS CALL Routines under OS/2

SAS System CALL routines are used to alter variable values or perform other system functions. Most CALL routines are completely described in the functions and CALL routines portion of *SAS Language Reference: Dictionary*. The CALL routines that are described here have syntax or behavior specific to the OS/2 operating environment.

BYTE

Returns one character in the ASCII collating sequence

OS/2 specifics: Uses the ASCII code sequence

Syntax

BYTE(*n*)

n
: is an integer that specifies which character in the ASCII collating sequence to return. The value of *n* can range from 0 to 255.

Details Because OS/2 is an ASCII system, the BYTE function returns the *n*th character in the ASCII collating sequence. The value of *n* can range from 0 to 255.

Any programs using the BYTE function with characters above ASCII 127 (the hexadecimal notation is **`'7F'x`**) may return a different value when used on a PC from another country as characters above ASCII 127 are national characters and they vary from country to country.

See Also

- BYTE function in *SAS Language Reference: Dictionary*

CALL DMYTECKS

Calculates the checksum (exclusive OR) of all the characters in a DataMyte packet, excluding the checksum itself

OS/2 specifics all

Syntax

DMYTECKS (*string*, *initial-cks*, *calculated-cks*)

string
: is the string for which the checksum is calculated. This argument can be a character variable, a character literal enclosed in quotation marks, or another character expression. A DataMyte can transmit a string of up to 256 characters.

initial-cks
: is the initial checksum value. This value is ”00’x if the string is under 200 characters.

calculated-cks
: is the calculated checksum for *string*. This value is a 2–byte hexadecimal number.

Details

The CALL DMYTECKS routine calculates the checksum for a packet. The checksum is the exclusive OR (XOR) of all the characters in the packet (including the start-of-text character, the character count, and end-of-transmission character), excluding the checksum itself. Because the length of SAS character variables is limited to 200, if you want to calculate the checksum for a packet that is longer than 200 characters, you have to call the CALL DYMTECKS routine twice. Call it once for the first 200 characters of the packet. Then pass the remaining characters to the CALL DYMTECKS routine using the calculated checksum from the first call as the initial checksum value for the second call.

Example

In the following example, the final checksum is stored in CALC_CS2.

```
data _null_;
   length string1 string2 $200 checksm1 checksm2 calc_cs1 calc_cs2;
     /* The string received from the DataMyte is longer than 200     */
     /* characters, so it is split into two string, STRING1 and      */
     /* STRING2. However, you must calculate the checksum for the    */
     /* entire string (STRING1||STRING2).  SAS statements reading in */
     /* data from DataMyte until EOT is found.                       */
     /* Initialize the first checksum.                               */
   checksm1='00'x;
   call dmytecks(string1,ckecksm1,calc_cs1);
   checksm2=calc_cs1;
   call dmytecks(string2,checksm2,calc_cs2);
run;
```

See Also

□ "Reading Data Using DataMyte Processing" on page 100

CALL SOUND

Generates a sound with a specific frequency and duration

OS/2 specifics: all

Syntax

CALL SOUND(*frequency*,*duration*)

frequency
specifies the sound frequency in terms of cycles per second. The frequency must be at least 20 and no greater than 20,000.

duration
specifies the sound duration in 1/80ths of a second.

Example

Example 1: Producing a Tone The following statement produces a tone of frequency 523 cycles per second (middle C) lasting 2 seconds:

```
data _null_;
   call sound(523,160);
run;
```

CALL SYSTEM

Issues operating environment commands

OS/2 specifics: *command* must be a valid OS/2 command

Syntax

CALL SYSTEM(*command*)

command

can be any of the following:

- □ an operating environment command enclosed in quotes or the name of a OS/2 application that is enclosed in quotes
- □ an expression whose value is an operating environment command or the name of a OS/2 application
- □ the name of a character variable whose value is an operating environment command or the name of a OS/2 application.

Details If you are running SAS interactively and the command that you run is an OS/2-based command or program, the command executes in a command prompt window. By default, you must type **`exit`** to return to your SAS session.

Comparison The CALL SYSTEM routine is similar to the X command. However, the CALL SYSTEM routine is callable and can therefore be executed conditionally. An example of using the CALL SYSTEM routine is given in “Executing Operating Environment Commands Conditionally” on page 21.

The values of the XSYNC and XWAIT system options affect how the CALL SYSTEM routine works. For more information about these options, see “XSYNC” on page 386 and “XWAIT” on page 387.

Examples

Example 1: Executing Operating System Commands Conditionally If you want to execute operating environment commands conditionally, use the CALL SYSTEM routine:

```
options noxwait;
data _null_;
   input flag $ name $8.;
   if upcase(flag)='Y' then
      do;
         command='md c:\'||name;
         call system(command);
      end;
   cards;
Y mydir
Y junk2
N mydir2
Y xyz
;
```

This example uses the value of the variable FLAG to conditionally create directories. After the DATA step executes, three directories have been created: C:\MYDIR, C:\JUNK2, and C:\XYZ. The directory C:\MYDIR2 is not created because the value of FLAG for that observation is not **Y**.

The X command is a global SAS statement. Therefore, it is important to realize that you cannot conditionally execute the X command. For example, if you submit the following code, the X statement is executed:

```
data _null_;
   answer='n';
   if upcase(answer)='y' then
      do;
         x 'md c:\extra';
      end;
run;
```

In this case, the directory C:\EXTRA is created regardless of whether the value of ANSWER is equal to **'n'** or **'y'**.

Example 2: Obtaining a Directory Listing The following is an example of using the CALL SYSTEM routine to obtain a directory listing:

```
data _null_;
   call system('dir /w');
run;
```

In this example, the /W option for the DIR command instructs OS/2 to print the directory in the wide format instead of a vertical list format.

See Also

- CALL SYSTEM routine in *SAS Language Reference: Dictionary*
- Command: “X” on page 232
- System option: “XSYNC” on page 386
- System option: “XWAIT” on page 387

COLLATE

Generates a collating sequence character string

OS/2 specifics: Uses the ASCII code sequence

Syntax

COLLATE (*start-position*<,*end-position*>)

COLLATE(*start-position*<,,*length*>)

start-position
specifies the numeric position in the collating sequence of the first character to be returned.

end-position
specifies the numeric position in the collating sequence of the last character to be returned.

length
specifies the number of characters you want (the length of the returned string).

Details The COLLATE function returns a string of ASCII characters that range in value from 0 to 255. The string returned by the COLLATE function begins with the ASCII character specified by the *start-position* argument. If the *end-position* argument is specified, the string returned by the COLLATE function contains all the ASCII characters between the *start-position* and *end-position* arguments. If the *length* argument is specified instead of the *end-position* argument, then the COLLATE function returns a string with a length of *length*. The returned string ends, or truncates, with the character having the value 255 if you request a string length that contains characters exceeding this value.

If you assign the return value of the COLLATE function to a variable with a length less than 256, the ASCII collating sequence string is padded with blanks to a length of 256. If you request a length of more than 256 characters, the returned string is padded with blanks to a length of *length*.

Note: Any programs using the COLLATE function with characters above ASCII 127 (the hexadecimal notation is **'7F'x**) may return a different value when used on a PC from another country as characters above ASCII 127 are national characters and they vary from country to country. △

See Also

- COLLATE function in *SAS Language Reference: Dictionary*

DMYTECHC

Calculates the character count for a DataMyte packet

OS/2 specifics: all

Syntax

DMYTECHC ('*string*')

string
specifies a text string.

Details

The DMYTECHC function accepts a text string as an argument and returns the character count for that string. This count represents the number of characters in the packet, excluding the STX (start-of-transmission) character, the character count itself, and the checksum. The return value is a 2–byte, hexadecimal number.

The following example of using the DMYTECHC function also uses the DMYTECKS CALL routine, which is described later in this chapter.

```
data dmyte1;
   length cs calccs $2;
   cs='00'x;
   /* This is the start-of-transmission character */
   stx='02'x;
   /* This is the end-of-transmission character   */
   eot='04'x;
   /* The character count includes the text       */
   /* string and the EOT.                         */
   cmd='?SETUP'||eot;
   cc=dmytechc(cmd);
   put cc=;
   str=stx||cc||cmd;
   /* Call DMYTECKS to determine the checksum     */
   /* (CALCCS) for the packet.                    */
   call dmytecks(str,cs,calccs);
   put calccs=;
   dmytecmd=str||calccs;
run;
```

Next, you would send the value of the DMYTECMD variable to the DataMyte machine via the communications port.

See Also

- "Reading Data Using DataMyte Processing" on page 100

DMYTECWD

Determines the total number of words in a DataMyte packet

OS/2 specifics: all

Syntax

DMYTECWD (*first-string*, *second-string*)

first-string
represents a fewer-than-200 character portion of the packet.

second-string
represents the balance of the packet. If the packet is 200 or fewer characters long, specify *second-string* as a null string ('').

Details

The DMYTECWD function return the number of tokens, or words, in a packet. Because data packets sent from DataMyte can be up to 256 characters long, but the SAS System can only process strings up to 200 characters long, you may have to break the packet up into two strings. Always break up the packet at a token delimiter, which is a semicolon (;). Both the *first-string* and *second-string* arguments can be a character variable, a character literal enclosed in quotes, or another character expression.

The following example counts words in the two character variables, FIRSTSTR and STR.

```
data dmyte2;
   length firststr str $ 200;
   firststr='07/16/83,14:00;M. Jones;Press 1;000;2.43;2.91';
   str='2.83;2.80;';
   count=dymtecwd(firststr,str);
   /* This sample packet contains 18 words.   */
   put count=;
run;
```

See Also

"Reading Data Using DataMyte Processing" on page 100

DMYTERVC

Converts the DataMyte character count to an ASCII number

OS/2 specifics: all

Syntax

DMYTERVC (*hex-number*)

hex-number
specifies the 2–byte DataMyte character count in hexadecimal that is to be converted to anASCII number.

Details

The DMYTERVC functions helps you convert a DataMyte character count, which is a 2–byte hexadecimal number, to an ASCII number. This number represents the number of characters that DataMyte is transmitting. this number does not include the STX (start-of-transmission) character, the 2–byte character count characters, or the 2–byte checksum.

In the following example, the DMYTERVC function calculates the character count. Once the character count is know, it can be used to process the incoming data, such as separating the data into words and store those words in SAS variables.

```
data dmyte3;
   /* this is the start-of-transmission character       */
   stx='02'x;
   infile 'com1:' lrecl=1 recfm=f;
   input x $char1.;
   if x eq stx then
     do;
      input cc $char2.;
      datacnt=dmytervc(cc);
     end;
   /* The character count tells us how many characters  */
   /* are in the packet being sent from the data        */
   /* collector (DATANCT is number of characters        */
   /* calculated by the DMYTERVC function.              */
   do i = 1 to datacnt;
     index+1;
     input x $char1.;
     substr(str,index,1)=x;
   /* ...more data processing statements                */
   end;
run;
```

Some type of data you could expect in the packet include the data and time, identification information such as the name of the operator, and data values.

See Also

"Reading Data Using DataMyte Processing" on page 100

MCIPISLP

Causes the SAS System to wait for a piece of multimedia equipment to become active

OS/2 specifics: all

Syntax

rc=**MCIPISLP**(*number-of-seconds*)

rc
return code.

number-of-seconds
specifies the number of seconds you want the SAS System to wait. This number must be an integer.

Details The MCIPISLP function is especially useful when you have used the MCIPISTR function to open a piece of equipment, but you know it is going to take a few seconds for the equipment to be ready.

The *number-of-seconds* argument must be an integer and represents how many seconds you want to wait. The return value is the number of seconds slept.

The MCIPISLP function can be used in the DATA step and in SCL code.

Example

This example uses both the MCIPISTR and MCIPISLP functions to play a CD and a video. The PUT statements display the return values of these functions; this allows you to see in the SAS log whether there was a problem with any of your equipment:

```
data _null_;
  /* Open a CD player. */
  msg=mcipistr("open cdaudio alias mytunes");
  put msg=;
  /* Wait one second for the CD player     */
  /* to become active.                     */
  slept=mcipislp(1);
  /* Begin playing your favorite tunes     */
  /* from the beginning of the CD.         */
  msg=mcipistr("play mytunes");
  put msg=;
  /* Now open a video file. */
  msg=mcipistr("open c:\movies\amovie.avs
                alias myshow");
  put msg=;
  /* Begin the show and wait for it to     */
  /* complete.                             */
  msg=mcipistr("play myshow wait");
  put msg=;
  /* When the show is complete,            */
  /* close the instance.                   */
  msg=mcipistr("close myshow");
  put msg=;
  /* Stop and close the instance of the CD */
  /* player.                               */
  msg=mcipistr("stop mytunes");
  put msg=;
  msg=mcipistr("close mytunes");
  put msg=;
run;
```

See Also

- Function: “MCIPISTR” on page 254

MCIPISTR

Submits an MCI string command to a piece of multimedia equipment

OS/2 specifics: all

Syntax

rc=**MCIPISTR**(*MCI-string-command*)

rc
return code.

MCI-string-command
is any valid SAS string; that is, a character variable, a character literal enclosed in quotes, or other character expression.

Details The MCIPISTR function submits an MCI (Media Control Interface) string command.

You can use MCI to control many types of multimedia equipment, such as CD players, mixers, videodisc players, and so on. OS/2 provides MCI support. For more information about valid MCI string commands, refer to your MCI-compliant device documentation.

The return value is a string that contains return information from the MCI string command. Examples of return information include "invalid instance" and "1".

Note: Not all MCI commands supply return codes that are usable from the SAS System △

The MCIPISTR function can be used in the DATA step and in SCL code.

Example

In this example, to use a CD player you would submit the following statements in your DATA step:

```
msg=mcipistr("open cdaudio alias cd");
msg=mcipistr("play cd");
msg=mcipistr("stop cd");
msg=mcipistr("close cd");
```

See Also

- Function: “MCIPISLP” on page 253

MODULE*xy*

Calls a specific routine or module that resides in an external dynamic link library (DLL)

OS/2 specifics: all

Syntax

CALL MODULE(<*cntl*>,*module*,*arg-1*,*arg-2*. . . ,*arg-n*);

num=**MODULEN**(<*cntl*>,*module*,*arg-1*,*arg-2*…,*arg-n*);

char=**MODULEC**(<*cntl*>,*module*,*arg-1*…,*arg-2*,*arg-n*);

Note: The following functions permit vector and matrix arguments; you can use them within the IML procedure. △

CALL MODULEI <*cntl*>,*modulearg-1*,*arg-2*. . . ,*arg-n*);

num=**MODULEIN**(<*cntl*>,*module*,*arg-1*,*arg-2*. . .,*arg-n*) *char*=**MODULEIC**(<*cntl*>,*module*,*arg-1*,*arg-2*. . .,*arg-n*);

cntl

is an optional control string whose first character must be an asterisk (*), followed by any combination of the following characters:

I	prints the hexadecimal representations of all arguments to the MODULE*xy* function and to the requested DLL routine before and after the DLL routine is called. You can use this option to help diagnose problems that are caused by incorrect arguments or attribute tables. If you specify the **`I`** option, the **`E`** option is implied.
E	prints detailed error messages. Without the **`E`** option (or the **`I`** option, which supersedes it), the only error message that the MODULE*xy* function generates is "Invalid argument to function," which is usually not enough information to determine the cause of the error.
S*x*	uses *x* as a separator character to separate field definitions. You can then specify *x* in the argument list as its own character argument to serve as a delimiter for a list of arguments that you want to group together as a single structure. Use this option only if you do not supply an entry in the SASCBTBL attribute table. If you do supply an entry for this module in the SASCBTBL attribute table, you should use the FDSTART option in the ARG statement in the table to separate structures.
H	provides brief help information about the syntax of the MODULE*xy* routines, the attribute file format, and the suggested SAS formats and informats.

For example, the control string **`'*IS/'`** specifies that parameter lists be printed and that the string **`'/'`** is to be treated as a separator character in the argument list.

module

is the name of the external module to use, specified as a DLL name and the routine name or ordinal value, separated by a comma. The module must reside in a dynamic link library (DLL) and it must be externally callable. For example, the value **`'DOSCALLS,230'`** specifies to load DOSCALLS.DLL and to invoke the routine identified by ordinal 230. Note that although the DLL name is not case sensitive, the routine name is based on the restraints of the routine's implementation language, so the routine name is case sensitive.

Note: DOSCALLS.DLL is an internal DLL provided by OS/2; you cannot find it by searching your disk. However, its routines are available for your use. △

If the DLL supports ordinal-value naming, you can provide the DLL name followed by a decimal number, such as **`'XYZ,30'`**.

You do not need to specify the DLL name if you specified the MODULE attribute for the routine in the SASCBTBL attribute table, as long as the routine name is unique (that is, no other routines have the same name in the attribute file).

You can specify *module* as a SAS character expression instead of as a constant; most often, though, you will pass it as a constant.

arg-1, arg-2, ...arg-n

are the arguments to pass to the requested routine. Use the proper attributes for the arguments (that is, numeric arguments for numeric attributes and character arguments for character attributes).

CAUTION:

Be sure to use the correct arguments and attributes. If you use incorrect arguments or attributes for a DLL function, you can cause the SAS environment, and possibly your operating environment, to fail. △

Details The MODULE functions execute a routine *module* that resides in an external (outside the SAS System) dynamic link library with the specified arguments *arg-1* through *arg-n*.

The MODULE call routine does not return a value, while the MODULEN and MODULEC functions return a number *num* or a character *char*, respectively. Which routine you use depends on the expected return value of the DLL function you want to execute.

MODULEI, MODULEIC, and MODULEIN are special versions of the MODULE*xy* functions that permit vector and matrix arguments. Their return values are still scalar. You can invoke these functions only from PROC IML.

Other than this name difference, the syntax for all six routines is the same.

The MODULE*xy* function builds a parameter list by using the information in *arg-1* to *arg-n* and by using a routine description and argument attribute table that you define in a separate file. Before you invoke the MODULE*xy* routine, you must define the fileref of SASCBTBL to point to this external file. You can name the file whatever you want when you create it.

This way, you can use SAS variables and formats as arguments to the MODULE*xy* function and ensure that these arguments are properly converted before being passed to the DLL routine.

See Also

- "The SASCBTBL Attribute Table" on page 166

PEEK

Accesses the data stored in a specific location in memory

OS/2 specifics: all

Syntax

data=**PEEKC**(*address*,*length*);

data=**PEEK**(*address*,*length*);

data
is the value that is returned by the function.

address
specifies the name identifying a location (address) in memory.

length
specifies the length of the returned value.

Details

CAUTION:

Use the PEEK functions only to access information returned by one of the MODULE*xy* functions. The PEEK functions can directly access memory addresses. Improper use of these functions can cause the SAS System, and your operating environment, to fail. △

The PEEK function returns to *data* a value of length *length* that contains the data that start at memory address *address*.

The variations of the PEEK functions are:

PEEKC	accesses character strings.
PEEK	accesses numeric values.

Usually, when you need to use one of the PEEK functions, you will use PEEKC to access a character string. The PEEK function is mentioned here for completeness.

RANK

Returns the position of a character in the ASCII collating sequence

OS/2 specifics: Uses the ASCII collating sequence

Syntax

RANK(*x*)

x

is a character in the ASCII collating sequence.

Details Because OS/2 is an ASCII system, the RANK function returns an integer that represents the position of a character in the ASCII collating sequence. The *x* argument must represent a character in the ASCII collating sequence. If the length of *x* is greater than 1, you receive the rank of the first character in the string.

Note: Any program that uses the RANK function with characters above ASCII 127 (the hexadecimal notation is **'7F'x**) is not portable because these are national characters and they vary from country to country. △

See Also

- □ RANK function in *SAS Language Reference: Dictionary*

SLEEP

Suspends execution of a SAS DATA step for a specified number of seconds

OS/2 specifics: all

Syntax

SLEEP(*num-seconds*)

num-seconds
specifies the number of seconds you want to suspend execution of a DATA step. The *num–seconds* argument is a numeric constant that must be greater than or equal to 0. Negative or missing values for *num–seconds* are invalid.

Details The SLEEP function is useful for scheduling tasks, such as collecting data from the communications port.

The return value of the *num–seconds* argument is the number of seconds slept. The maximum sleep period for the SLEEP function is approximately 46 days.

When you submit a program that calls the SLEEP function, a pop-up window appears telling you how long the SAS System is going to sleep. Your SAS session remains inactive until the sleep period is over. If you want to cancel the call to the SLEEP function, use the CTRL+BREAK attention sequence.

You should use a null DATA step to call the SLEEP function; follow this DATA step with the rest of the SAS program. Using the SLEEP function in this manner enables you to use the CTRL+BREAK attention sequence to interrupt the SLEEP function and to continue with the execution of the rest of your SAS program.

Example

This example of the SLEEP function tells the SAS System to delay the execution of the program for 12 hours and 15 minutes:

```
data _null_;
   /* argument to sleep must be expressed in seconds */
   slept=sleep((60*60*12)+(60*15));
run;
data monthly;
   /*... more data lines */
run;
```

TRANSLATE

Replaces specific characters in a character expression

OS/2 specifics: Required syntax; pairs of *to* and *from* arguments are optional

Syntax

TRANSLATE(*source,to-1,from-1 <…to-n,from-n>*)

source
specifies the SAS expression containing the original character value.

to
specifies the characters you want TRANSLATE to use as substitutes.

from
specifies the characters you want TRANSLATE to replace.

Details Under OS/2, you do not have to provide pairs of *to* and *from* arguments. However, if you do not use pairs, you must supply a comma as a place holder.

See Also

- TRANSLATE function in *SAS Language Reference: Dictionary*

WAKEUP

Specifies the time a SAS DATA step begins execution

OS/2 specifics: all

Syntax

WAKEUP(*until-when*)

until-when
specifies the time when the WAKEUP function will be executed.

Details Use the WAKEUP function to specify the time a DATA step begins to execute. The return value is the number of seconds slept.

The *until-when* argument can be a SAS datetime value, a SAS time value, or a numeric constant, as explained in the following list:

- If *until-when* is a datetime value, the WAKEUP function sleeps until the specified date and time. If the specified date and time have already passed, the WAKEUP function does not sleep, and the return value is 0.
- If *until-when* is a time value, the WAKEUP function sleeps until the specified time. If the specified time has already passed in that 24-hour period, the WAKEUP function sleeps until the specified time occurs again.
- If the value of *until-when* is a numeric constant, the WAKEUP function sleeps for that many seconds before or after the next occurring midnight. If the value of *until-when* is a positive numeric constant, the WAKEUP function sleeps for *until-when* seconds past midnight. If the value of *until-when* is a negative numeric constant, the WAKEUP function sleeps until *until-when* seconds before midnight.

Negative values for the *until-when* argument are allowed, but missing values are not. The maximum sleep period for the WAKEUP function is approximately 46 days.

When you submit a program that calls the WAKEUP function, a pop-up window appears telling you when the SAS System is going to wake up. Your SAS session remains inactive until the waiting period is over. If you want to cancel the call to the WAKEUP function, use the CTRL+BREAK attention sequence.

You should use a null DATA step to call the WAKEUP function; follow this DATA step with the rest of the SAS program. Using the WAKEUP function in this manner enables you to use the CTRL+BREAK attention sequence to interrupt the waiting period and continue with the execution of the rest of your SAS program.

Examples

Example 1: Delaying Program Execution until a Specified Date or Time The following example tells the SAS System to delay execution of the program until 1:00 p.m. on January 1, 1999:

```
data _null_;
   slept=wakeup('01JAN1999:13:00:00'dt);
run;
data compare;
   /* ...more data lines */
run;
```

The following example tells the SAS System to delay execution of the program until 10:00 p.m.:

```
data _null_;
   slept=wakeup("22:00:00"t);
run;
data compare;
   /* ...more data lines */
run;
```

Example 2: Delaying Program Execution until a Specified Time Period after Midnight This example tells the SAS System to delay execution of the program until 35 seconds after the next occurring midnight:

```
data _null_;
   slept=wakeup(35);
run;
data compare;
   /* ...more data lines */
run;
```

Example 3: Using a Variable as an Argument to the WAKEUP Function This example illustrates using a variable as the argument of the WAKEUP function:

```
data _null_;
   input x;
   slept=wakeup(x);
   cards;
1000
;
data compare;
   input article1 $ article2 $ rating;
   /* ...more data lines */
run;
```

Because the instream data indicate that the value of X is 1000, the WAKEUP function sleeps for 1,000 seconds past midnight.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Companion for the OS/2® Environment, Version 8*, Cary, NC: SAS Institute Inc., 1999. 448 pp.

SAS® Companion for the OS/2® Environment, Version 8

ISBN 1-58025-521-3

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

1st printing, October 1999