Bugs Affecting SUDAAN 10.0.1

This list was updated on August 26, 2013. The most recent list can be found at http://www.rti.org/sudaan/page.cfm/Known_Bugs.

277. LOGISTIC (RLOGIST), LOGLINK, MULTILOG, and REGRESS Procedures: Missing or incorrect labels and headers when a single continuous variable is specified on the PREDMARG statement.

300. HOTDECK, KAPMEIER, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, REGRESS, WTADJUST, WTADJX Procedures: SUDAAN may produce corrupted output files for FLAT output groups.

301. HOTDECK Procedure: Character variables on the IDVAR statement are not handled correctly

302. ALL Procedures: PROGRAMMER ERROR when a double hyphen ('--') is used to specify a sequential set of variables from the input file.

303. LOGISTIC (RLOGIST) and MULTILOG Procedures: PROGRAMMER ERROR when the ADJRR option is present on the PREDMARG or CONDMARG statements, and a categorical variable on the statement has only 1 level.

304. LOGISTIC (RLOGIST) and MULTILOG Procedures: Incorrect PRED_SERR, PRED_LOWRR, and PRED_UPRR statistics for Delete-1 Jackknife designs when virtual memory needed

305. LOGISTIC (RLOGIST) and MULTILOG Procedures: PROGRAMMER ERROR with the ADJRR option on the CONDMARG statement for Delete-1 Jackknife designs when virtual memory required

306. MULTILOG and REGRESS Procedures: Incorrect VARPRMG statistic for Delete-1 Jackknife designs when virtual memory not required

307. REGRESS Procedure: PROGRAMMER ERROR with the CONDMARG and LSMEANS statements for Delete-1 Jackknife designs when virtual memory required

308. CROSSTAB Procedure: Incorrect values for MHOR, MHRR1, and MHRR2 when the table includes cells with a weighted frequency of zero

309. CROSSTAB Procedure: SEMANTIC ERROR with the GOFIT statement and INCLUDE=NONMISSING on the SUBGROUP statement

310. CROSSTAB, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, REGRESS and WTADJUST Procedures: PROGRAMMER ERROR and unexpected SEMANTIC ERROR when using the NOTSORTED option

312. CROSSTAB Procedure: A PROGRAMMER ERROR may occur when a variable on the TABLES statement is not specified as categorical.

277. LOGISTIC (RLOGIST), LOGLINK, MULTILOG, and REGRESS Procedures: Missing or incorrect labels and headers when a single continuous variable is specified on the PREDMARG statement.

Description:

This bug relates to the PREDMARG (but not CONDMARG) statement in the REGRESS, LOGISTIC (RLOGIST), LOGLINK, and MULTILOG (but not SURVIVAL) procedures. When there is a single continuous variable specified, the table header and variable label on the printed table are missing or incorrect. The variable name and variable label on the output dataset are also incorrect. The values of the computed marginals are correct, however.

Work-around:

A partial work-around for this problem is to add INTERCEPT or another variable to the PREDMARG statement.

Example:

In the code below, the first PREDMARG statement produces a table which illustrates the problem, and the second PREDMARG statement produces a table with appropriate labels.

```
  WEIGHT wgt;
  CLASS x05 x06 x07 x08;
  MODEL LMEAS = X01 x02 x03 x04 x05 x06 x07 x08;
  PREDMARG X01 / X01=(.01);
  PREDMARG INTERCEPT X01 / X01=(.01);
  OUTPUT PREDMARG / FILENAME=PM;
```

Output from the first PREDMARG statement displays the label for the WEIGHT variable (wgt) instead of the continuous covariate X01. In addition, several lines of information are missing from the table header.

```
by: Variable.

-------------------------------------------------------------------------
Variable               Predicted
                       Marginal             SE     T:Marg=0    P-value
-------------------------------------------------------------------------
Full Sample 6 Year
  MEC Exam Weight            0.13         0.00        53.54     0.0000
-------------------------------------------------------------------------
```

The PM dataset created by the OUTPUT statement includes a variable named VARIABLE with the label "Variable". It should, however, be named PREDMARG1 with the label "Predicted Marginal #1".

Output from the second PREDMARG statement displays the correct labels and header:

```
Variance Estimation Method: Taylor Series (WR)
SE Method: Robust (Binder, 1983)
Working Correlations: Independent
Link Function: Identity
```

```
Response variable LMEAS: LMEAS
by: Predicted Marginal #2.

------------------------------------------------------------------------
Predicted Marginal     Predicted
  #2                   Marginal             SE     T:Marg=0    P-value
------------------------------------------------------------------------
Intercept                    0.13         0.00        53.81     0.0000
X01
    0.0100                   0.13         0.00        53.54     0.0000
------------------------------------------------------------------------
```

The PM dataset correctly includes a variable named PREDMARG2 with the label "Predicted Marginal #2".

System	Release Reported	Release Fixed
Windows	9.0.3	11.0.1
Solaris	9.0.2	11.0.0
Linux	9.0.2	11.0.0

300. HOTDECK, KAPMEIER, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, REGRESS, WTADJUST, WTADJX Procedures: SUDAAN may produce corrupted output files for FLAT output groups.

Description:

When a FLAT output group is used, such as the PREDICTED group in the REGRESS procedure or the KAPMEIER group in the KAPMEIER procedure, and the output file type is not SAS, the output file may be corrupted. This can lead to an abnormal termination of SAS when the file is read in later.

Work-around:

Use FILETYPE=SAS for the output file, then use the RECORDS procedure to convert the file to the desired type.

Example:

The following program produces a corrupted output file, and SAS terminates abnormally when the RECORDS procedure is run:

```
proc hotdeck data=wicdat filetype=sudxport seed=66187573 notsorted;
    weight analwgt1;
    impby momrace momsmk moincom marital2;
    impcond babywgt>=0 and momhosp>=0;
    impvar babywgt momhosp;
    impname babywgt="bwgt_i" momhosp="mhosp_i";
    impid ID;
    idvar / all;
    output / impute=all filename=out1.sdx filetype=sudxport replace;
run;

proc records data=out1.sdx filetype=sudxport contents;
    print / maxrec=25;
run;
```

Changing the FILETYPE to SAS as shown below produces the desired output file with no errors:

```
proc hotdeck data=wicdat filetype=sudxport seed=66187573 notsorted;
    weight analwgt1;
    impby momrace momsmk moincom marital2;
    impcond babywgt>=0 and momhosp>=0;
    impvar babywgt momhosp;
    impname babywgt="bwgt_i" momhosp="mhosp_i";
    impid ID;
    idvar / all;
    output / impute=all filename=out1 filetype=sas replace;
run;

proc records data=out1 filetype=sas contents;
    print / maxrec=25;
    output / filename=out1.sdx filetype=sudxport replace;
run;
```

System	Release Reported	Release Fixed
Windows	10.0.0	11.0.0
Solaris	10.0.0	11.0.0
Linux	10.0.0	11.0.0

301. HOTDECK Procedure: Character variables on the IDVAR statement are not handled correctly

Description:

When a character variable is included on the IDVAR statement in the HOTDECK procedure, SUDAAN does not store the character variable on the output dataset correctly. As a result, attempts to read the dataset produce unexpected results, such as the following:

- Reading the dataset may produce a PROGRAMMER ERROR.
- Reading the dataset may crash SUDAAN and/or SAS.
- The character variable may be recognized as numeric.
- The character variable's value may be stored as blank on every record.

No other procedure in SUDAAN permits a character variable to be included on the IDVAR statement, so this bug applies only to HOTDECK.

Work-around:

Use only numeric variables on the IDVAR statement.

Example:

Suppose LASTNAME is a character variable on the INDATA dataset, and you submit the following HOTDECK call:

```
proc hotdeck data=indata seed=1;
    weight wt;
    impby ic;
    impvar impvar;
    idvar lastname;
    output / impute=default filename=outdata replace;
run;
```

The LASTNAME variable will be saved to the OUTDATA dataset, but reading the dataset may cause fatal errors. Even if the dataset can be read without error, the values of the LASTNAME variable will be incorrect.

System	Release Reported	Release Fixed
Windows	10.0.0	11.0.0
Solaris	10.0.0	11.0.0
Linux	10.0.0	11.0.0

302. ALL Procedures: PROGRAMMER ERROR when a double hyphen ('--') is used to specify a sequential set of variables from the input file.

Description:

When the double hyphen ('--') is used to specify a sequential set of variables from the input file, SUDAAN sometimes produces a PROGRAMMER ERROR and halts. However, this is not an error; SUDAAN should continue processing the job.

Work-around:

Eliminate the double hyphen by explicitly listing each variable to be included or use the single hyphen notation.

Example:

Running the following program in SUDAAN will result in a PROGRAMMER ERROR because the double hyphen is used in specifying the variables on the MODEL statement:

```
PROC LOGISTIC DATA=mydata DESIGN=WR;
NEST      stratum nfsu / missunit;
WEIGHT    FINALWT;
MODEL     resp = var1 -- var5 / noint;
```

To avoid the error, list the individual variables in the MODEL statement:

```
PROC LOGISTIC DATA=mydata DESIGN=WR;
NEST      stratum nfsu / missunit;
WEIGHT    FINALWT;
MODEL     resp = var1 var2 var3 var4 var5 / noint;
```

Or use the single hyphen:

```
PROC LOGISTIC DATA=mydata DESIGN=WR;
NEST      stratum nfsu / missunit;
WEIGHT    FINALWT;
MODEL     resp = var1 - var5 / noint;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

303. LOGISTIC (RLOGIST) and MULTILOG Procedures: PROGRAMMER ERROR when the ADJRR option is present on the PREDMARG or CONDMARG statements, and a categorical variable on the statement has only 1 level.

Description:

When the ADJRR option is present on the PREDMARG or CONDMARG statement and a categorical variable on the statement has only 1 level, SUDAAN cannot complete the calculations. In this situation, it should halt with an appropriate error message. However, it issues a PROGRAMMER ERROR instead.

Work-around:

Remove the ADJRR option from the PREDMARG or CONDMARG statement when there is a variable with only 1 level on the statement.

Example:

Running the following program in SUDAAN will result in a PROGRAMMER ERROR because the variable DUMMY has only one level:

```
PROC LOGISTIC DATA=mydata DESIGN=WR;
NEST      stratum nfsu / missunit;
WEIGHT    FINALWT;
SUBGROUP  dummy gender;
LEVELS        1      2;
MODEL     resp = dummy gender / noint;
PREDMARG  dummy gender / adjrr;
```

To avoid the error, remove the variable DUMMY from the PREDMARG statement:

```
PROC LOGISTIC DATA=mydata DESIGN=WR;
NEST      stratum nfsu / missunit;
WEIGHT    FINALWT;
SUBGROUP  dummy gender;
LEVELS        1      2;
MODEL     resp = dummy gender / noint;
/* remove DUMMY from the PREDMARG statement */
PREDMARG  gender / adjrr;
```

Or remove the ADJRR option:

```
PROC LOGISTIC DATA=mydata DESIGN=WR;
NEST      stratum nfsu / missunit;
WEIGHT    FINALWT;
SUBGROUP  dummy gender;
LEVELS        1      2;
MODEL     resp = dummy gender / noint;
/* remove the ADJRR option */
```

```
PREDMARG  dummy gender;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

304. LOGISTIC (RLOGIST) and MULTILOG Procedures: Incorrect PRED_SERR, PRED_LOWRR, and PRED_UPRR statistics for Delete-1 Jackknife designs when virtual memory needed

Description:

The PRED_SERR, PRED_LOWRR, and PRED_UPRR statistics in PROC LOGISTIC and PROC MULTILOG are computed incorrectly when a Delete-1 Jackknife design is specified, virtual memory is needed to run the job, and physical memory will not hold the full set of replicate risk ratios.

The conditions for reproducing this bug are data-dependent, and therefore the bug will not appear in all analyses. The size of the dataset and the calculations requested will determine whether virtual memory is used. Alternatively, the user may force virtual memory to be used by specifying the USEVMEM=1 option on the PROC statement. Even if virtual memory is used, the full set of replicate risk ratios will fit into physical memory for some datasets but not for others.

Work-around:

Make enough physical memory (RAM) available to SUDAAN so that it does not need virtual memory to process the job. This may not be possible, however, depending on the size of the dataset and the calculations requested. Methods you can try include:

- Close other applications to free up RAM for use by SUDAAN.
- Use a computer with more RAM. Please note, however, that your operating system may place an upper limit on the amount of RAM available to a single process, no matter how much RAM the computer has. For example, 32-bit Windows has a 2 gigabyte limit.

When trying these methods, be sure not to use the USEVMEM=1 option.

Example:

The following program produces an incorrect standard error for the adjusted risk ratio, which leads to an incorrect confidence interval. In this example, the USEVMEM option was provided to demonstrate the bug by forcing the use of virtual memory, but the more likely scenario is that virtual memory is required because of the size of the dataset.

```
proc rlogist data=mydata design=jackknife usevmem=1;
    nest stratum psu;
    weight wt;
    subgroup gender race;
    levels 2 3;
    model pass = age gender race;
    predmarg race / adjrr;
    print / predrisk=all;
run;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

305. LOGISTIC (RLOGIST) and MULTILOG Procedures: PROGRAMMER ERROR with the ADJRR option on the CONDMARG statement for Delete-1 Jackknife designs when virtual memory required

Description:

SUDAAN produces a PROGRAMMER ERROR when the ADJRR option is given on the CONDMARG statement in PROC LOGISTIC or PROC MULTILOG, a Delete-1 Jackknife design is specified, and virtual memory is required to run the job.

The conditions for reproducing this bug are data-dependent, and therefore the bug may not appear in all analyses. The size of the dataset and the calculations requested will determine whether virtual memory is used. Alternatively, the user may force virtual memory to be used by specifying the USEVMEM=1 option on the PROC statement.

Work-around:

Make enough physical memory (RAM) available to SUDAAN so that it does not need virtual memory to process the job. This may not be possible, however, depending on the size of the dataset and the calculations requested. Methods you can try include:

- Close other applications to free up RAM for use by SUDAAN.
- Use a computer with more RAM. Please note, however, that your operating system may place an upper limit on the amount of RAM available to a single process, no matter how much RAM the computer has. For example, 32-bit Windows has a 2 gigabyte limit.

When trying these methods, be sure not to use the USEVMEM=1 option.

Example:

The following program produces a PROGRAMMER ERROR. In this example, the USEVMEM option was provided to demonstrate the bug by forcing the use of virtual memory, but the more likely scenario is that virtual memory is required because of the size of the dataset.

```
proc rlogist data=mydata design=jackknife usevmem=1;
    nest stratum psu;
    weight wt;
    subgroup gender race;
    levels 2 3;
    model pass = age gender race;
    condmarg race / adjrr;
    print / condrisk=all;
run;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

306. MULTILOG and REGRESS Procedures: Incorrect VARPRMG statistic for Delete-1 Jackknife designs when virtual memory not required

Description:

The VARPRMG statistic (and its related statistics: SEPRDMRG, PREDVAR, T_PRDMRG, and P_PRDMRG) in REGRESS and MULTILOG is computed incorrectly when a Delete-1 Jackknife design is specified, no statistics other than those related to predicted marginals appear on the PRINT or OUTPUT statement, and virtual memory is not used.

Specifying an additional statistic for PRINT or OUTPUT may result in a correct VARPRMG, depending on the statistic. For example, adding CONDMRG or SEBETA to the PRINT statement fixes the problem, while adding BETA or DDF does not.

The conditions for reproducing this bug are data-dependent, and therefore the bug will not appear in all analyses. The size of the dataset and the calculations requested will determine whether virtual memory is used. Alternatively, the user may force virtual memory to be used by specifying the USEVMEM=1 option on the PROC statement.

Work-around:

Add CONDMRG or SEBETA to the list of statistics on the PRINT or OUTPUT statements. Alternatively, set the USEVMEM parameter to 1 to force the use of virtual memory.

Example:

The following program produces incorrect values for the standard error, variance, t-statistic, and p-value of the predicted marginal. In this example, the USEVMEM parameter was set to 0 to demonstrate the bug by preventing the use of virtual memory, but the more likely scenario is that the USEVMEM parameter is omitted and the dataset is small enough not to require virtual memory.

```
proc regress data=mydata design=jackknife usevmem=0;
    nest stratum psu;
    weight wt;
    subgroup gender race;
    levels 2 3;
    model score = age gender race;
    predmarg race;
    print / pred_mrg=all;
run;
```

Adding the SEBETA keyword to the PRINT statement is a work-around for the bug:

```
    print sebeta / pred_mrg=all;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

307. REGRESS Procedure: PROGRAMMER ERROR with the CONDMARG and LSMEANS statements for Delete-1 Jackknife designs when virtual memory required

Description:

SUDAAN produces a PROGRAMMER ERROR when the CONDMARG or LSMEANS statement is included in PROC REGRESS, a Delete-1 Jackknife design is specified, and virtual memory is required to run the job.

The conditions for reproducing this bug are data-dependent, and therefore the bug may not appear in all analyses. The size of the dataset and the calculations requested will determine whether virtual memory is used. Alternatively, the user may force virtual memory to be used by specifying the USEVMEM=1 option on the PROC statement.

Work-around:

Make enough physical memory (RAM) available to SUDAAN so that it does not need virtual memory to process the job. This may not be possible, however, depending on the size of the dataset and the calculations requested. Methods you can try include:

- Close other applications to free up RAM for use by SUDAAN.
- Use a computer with more RAM. Please note, however, that your operating may place an upper limit on the amount of RAM available to a single process, no matter how much RAM the computer has. For example, 32-bit Windows has a 2 gigabyte limit.

When trying these methods, be sure not to use the USEVMEM=1 option.

Example:

The following program produces a PROGRAMMER ERROR. In this example, the USEVMEM option was provided to demonstrate the bug by forcing the use of virtual memory, but the more likely scenario is that virtual memory is required because of the size of the dataset.

```
proc regress data=mydata design=jackknife usevmem=1;
    nest stratum psu;
    weight wt;
    subgroup gender race;
    levels 2 3;
    model pass = age gender race;
    condmarg race;
    print / cond_mrg=all;
run;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

308. CROSSTAB Procedure: Incorrect values for MHOR, MHRR1, and MHRR2 when the table includes cells with a weighted frequency of zero

Description:

When computing the MHOR (Mantel-Haenszel adjusted odds ratio) in CROSSTAB, SUDAAN drops a stratum if any of the weighted cell frequencies in that stratum is zero. However, it should not drop the stratum in that case. It should drop the stratum only if *all* weighted cell frequencies in that stratum are zero.

When computing the MHRR1 (Mantel-Haenszel adjusted risk ratio, column 1) in CROSSTAB, SUDAAN drops a stratum if the weighted frequency in either cell of the first column is zero or if the weighted frequency in either row is zero. Like the MHOR, it should drop the stratum only if all weighted cell frequencies in that stratum are zero.

When computing the MHRR2 (Mantel-Haenszel adjusted risk ratio, column 2) in CROSSTAB, SUDAAN drops a stratum if the weighted frequency in either cell of the second column is zero or if the weighted frequency in either row is zero. Like the MHOR, it should drop the stratum only if all weighted cell frequencies in that stratum are zero.

When all weighted cell frequencies in a stratum are zero, SUDAAN does appropriately drop the stratum from the calculation of MHOR, MHRR1, and MHRR2. However, it produces a warning message with misleading information. The message states:

> One or more 0 cells for VAR1 = 3. This stratum will be dropped from the calculation of MHOR.

when it should actually state:

> All cells are 0 for VAR1 = 3. This stratum will be dropped from the calculation of MHOR.

Work-around:

No work-arounds exist for this bug.

Example:

Consider an analysis of a 2x2x2 table with the following weighted cell frequencies:

$N_{111} = 1$	$N_{112} = 2$
$N_{121} = 1$	$N_{122} = 1$

$N_{211} = 1$	$N_{212} = 0$
$N_{221} = 1$	$N_{222} = 1$

From the SUDAAN manual, the MHOR formula for this example is:

$$(N_{111}N_{122}/N_{1++} + N_{211}N_{222}/N_{2++}) / (N_{112}N_{121}/N_{1++} + N_{212}N_{212}/N_{2++})$$

Using this formula, the MHOR should be:

(1*1/5 + 1*1/3) / (2*1/5 + 0*1/3) = 1.3

However, with the following program:

```
proc crosstab data=mydata design=srs;
    subgroup var1 var2 var3;
    levels 2 2 2;
    tables var1 * var2 * var3;
    risk mhor mhrr1 mhrr2;
    print wsum / adjrisk=all;
```

SUDAAN ignores the second stratum and computes the MHOR as:

(1*1/5) / (2*1/5) = 0.5

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

309. CROSSTAB Procedure: SEMANTIC ERROR with the GOFIT statement and INCLUDE=NONMISSING on the SUBGROUP statement

Description:

SUDAAN produces a semantic error if a PROC CROSSTAB job with the following set of characteristics is submitted:

- The GOFIT statement is specified, and
- A variable on the GOFIT statement also appears on the SUBGROUP statement, and
- The INCLUDE option is specified on the PROC statement, and
- The INCLUDE=NONMISSING parameter is specified on the SUBGROUP statement.

The text of the semantic error is:

> INCLUDE=MISSING not allowed on SUBGROUP/CLASS when a variable (*varname*) on one of these statements is also listed on the GOFIT statement.

However, a semantic error should not be produced under these conditions. The INCLUDE=NONMISSING parameter on the SUBGROUP statement should override the INCLUDE option on the PROC statement.

SUDAAN does not produce the semantic error when the CLASS statement is used instead of the SUBGROUP statement.

Work-around:

Specify the GOFIT variables on the CLASS statement instead of the SUBGROUP statement. If you use the INCLUDE option on the PROC statement (because you want to treat missing values as a legitimate level in most of your analysis variables), be sure to isolate the GOFIT variable on its own CLASS statement and specify the INCLUDE=NONMISSING parameter on that CLASS statement.

Example:

In the following program, the INCLUDE option was added to the PROC statement so that missing would be treated as a valid level in all categorical variables, but the INCLUDE=NONMISSING option was added to the SUBGROUP statement to override that for the SUBGROUP variables. The GOFIT variable appears on the SUBGROUP statement. This program meets all four criteria for the bug and therefore incorrectly produces a SEMANTIC ERROR.

```
proc crosstab data=mydata design=wr include;
    nest stratum psu;
    weight wt;
    subgroup educ gender / include=nonmissing;
    levels 3 2;
    class agecat race;
    gofit gender = 0.6 0.4;
    print / gof=all;
```

The work-around is to place the GENDER variable on a CLASS statement by itself:

```
proc crosstab data=mydata design=wr include;
    nest stratum psu;
    weight wt;
    subgroup educ / include=nonmissing;
    levels 3;
    class agecat race;
    class gender / include=nonmissing;
    gofit gender = 0.6 0.4;
    print / gof=all;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

310. CROSSTAB, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, REGRESS and WTADJUST Procedures: PROGRAMMER ERROR and unexpected SEMANTIC ERROR when using the NOTSORTED option

Description:

When using the NOTSORTED option, omitting a variable name from the CLASS or SUBGROUP statement may produce unexpected results in the CROSSTAB procedure or in the modeling procedures.

In CROSSTAB, the problem arises when a variable appears on the TABLES statement without also appearing on the CLASS or SUBGROUP statement. This should produce a SEMANTIC ERROR. However, SUDAAN produces a PROGRAMMER ERROR. This is true whether the variable is explicitly listed on the TABLES statement or it is implicitly listed using the dash or double-dash shortcut.

In the modeling procedures, the problem arises only for variables implicitly listed on the MODEL statement using the dash shortcut. If these variables do not appear on a CLASS or SUBGROUP statement, SUDAAN should treat them as continuous variables and run the analysis. Instead, it produces a SEMANTIC ERROR stating the variable is not available.

Work-around:

In PROC CROSSTAB, be sure that every variable you include on a TABLES statement is also included on either a CLASS or SUBGROUP statement.

In the modeling procedures, be sure that every categorical variable you include on a MODEL statement is also included on either a CLASS or SUBGROUP statement. In addition, list every continuous variable explicitly on the MODEL statement instead of using the dash shortcut. If listing every continuous variable proves difficult, another work-around is to sort your dataset prior to analyzing it with SUDAAN so that you can omit the NOTSORTED option. You could sort it either outside SUDAAN using other software or inside SUDAAN using the RECORDS procedure.

Examples:

The following programs produce a PROGRAMMER ERROR, but they should produce a SEMANTIC ERROR. For the second example, assume the GENDER variable appears on the input dataset between the AGE and RACE variables.

```
proc crosstab data=mydata design=wr notsorted;
    nest stratum psu;
    weight wt;
    class gender;
    tables race gender;
    print nsum wsum;

proc crosstab data=mydata design=wr notsorted;
    nest stratum psu;
    weight wt;
    class age race;
    tables age--race;
    print nsum wsum;
```

The following program produces a SEMANTIC ERROR stating that VISIT2 is not available, even though the dataset contains a variable named VISIT2.

```
proc rlogist data=mydata design=wr notsorted;
    nest stratum psu;
    weight wt;
    class visit1 visit3;
    model pass = visit1-visit3;
    print / betas=all;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

311. LOGISTIC (RLOGIST) and MULTILOG Procedures: There is a bug in the calculation of the confidence limits for the PREDMARG and CONDMARG risk ratios when the analysis is done using multiply imputed data.

Description:

The formula for the variance of the model-adjusted risk ratio (ADJRR option on PREDMARG and CONDMARG statements) using multiply imputed data (see Section 3.7.1 of the SUDAAN 10 Language Manual, Eq. 3.45) isn't being applied correctly.
The estimator in ***Equation 3.45*** is written as

$$\hat{V}_M\left(\hat{\theta}_M\right)=\bar{V}_m+\frac{m+1}{m}B_m,$$

where $\bar{V}_m$ is the mean of the variances over the imputations, and B_m is the variance of the imputed estimates.

Instead of using $\bar{V}_m$, SUDAAN is using $sqrt(\bar{V}_m)$.

Work-around:

The only work-around for this problem is to compute the variance $\hat{V}_M(\hat{\theta}_M)$ outside of SUDAAN and use that to compute the confidence limits.

Example:

The confidence limits will be incorrect for the program below, which uses MI data (MI_VAR statement) and computes the risk ratios on the PREDMARG statement:

```
proc rlogist data="..\Data\tempMI" filetype=sudxport;
  nest stratum psu;
  weight analwgt1;
  mi_var fam1-fam5;
  class racemom fam1;
  model BRFDINIT = racemom fam1;
  predmarg fam1 / adjrr;
  print / predrisk=all;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

312. CROSSTAB Procedure: A PROGRAMMER ERROR may occur when a variable on the TABLES statement is not specified as categorical.

Description:

The PROGRAMMER ERROR may be generated in PROC CROSSTAB when a variable on the TABLES statement is not specified as categorical by inclusion on the SUBGROUP or CLASS statement.

Work-around:

Use only categorical variables on the TABLES statement.

Example:

This is an example of a program that may produce a PROGRAMMER ERROR. Here, AGE is not specified as categorical on a SUBGROUP or CLASS statement.

```
proc crosstab data = "mydata" design=srs filetype=SUDXPORT;
tables age;
print nsum rowper serow;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

313. ALL Procedures: PROGRAMMER ERROR with the SAS TITLE*n* statement.

Description:

When one of the SAS TITLEn statements (TITLE1, TITLE2, etc.) appears within a SUDAAN procedure, SUDAAN may produce a PROGRAMMER ERROR.

Work-around:

To set SAS titles, move the SAS TITLE statements outside the SUDAAN procedure. To set SUDAAN titles, use the SUDAAN statement RTITLE. Multiple RTITLE statements may be included to create multi-line titles.

Example:

The following program may produce a PROGRAMMER ERROR:

```
proc records data=mydata;
    title "My Title";
    title2 "My Subtitle";
    run;
```

To use the titles as SAS titles, move them above the SUDAAN procedure:

```
title "My Title";
title2 "My Subtitle";
proc records data=mydata;
    run;
```

To use the titles as SUDAAN titles, use RTITLE:

```
proc records data=mydata;
    rtitle "My Title";
    rtitle "My Subtitle";
    run;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

314. ALL Procedures: Floating Point Error when a CLASS variable takes on very large floating point values

Description:

In cases where a CLASS variable takes on very large positive or negative values, SUDAAN may generate a floating point error as it attempts to create a label for the value. This occurs even with the NOFREQs option on the CLASS statement and even if the variable is not used in the program.

Work-around:

The only work around is to recode the variable outside of SUDAAN so that the values associated with the CLASS variable are within a normal integer range, or to remove the variable from the analysis.

Example:

Suppose the variable HUGE has the following distinct values on the input data set:

-99,888,777,666.555 43,149,618,112.567 56,789,123,456.789

SUDAAN will be unable to convert these values to labels for use in output tables, and will generate a floating point error. There is no way to recode these values within SUDAAN.

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

315. ALL Procedures: In 64-bit environments, SUDAAN may report no free disk space for use as virtual memory.

Description:

On some 64-bit Windows and Linux platforms, SUDAAN may report no free disk space for use as virtual memory and then halt. This may occur any time the NOTSORTED option is present, as well as in the REGRESS, LOGISTIC, LOGLINK, MULTILOG, and WTADJUST procedures.

Work-around:

The only work around for this problem is to include the option USEVMEM=0 on the PROC statement. This is viable as long as the computer has sufficient RAM to complete the analysis in memory without resorting to virtual disk space.

Example:

The following program using the REGRESS procedure could potentially generate the error on a 64-bit platform:

```
PROC REGRESS DATA=TEMP FILETYPE=SAS DESIGN=WR;
NEST _ONE_ DAMID;
WEIGHT_ONE_;
MODEL BW = DOSEGRP;
PRINT BETAS;
```

The workaround is to add USEVMEM=0 to the PROC statement:

```
PROC REGRESS DATA=TEMP FILETYPE=SAS DESIGN=WR USEVMEM=0;
NEST _ONE_ DAMID;
WEIGHT_ONE_;
MODEL BW = DOSEGRP;
PRINT BETAS;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

316. LOGISTIC (RLOGIST) Procedure: A PROGRAMMER ERROR occurs when the PSULEV option of the NEST statement is assigned an out-of-range value.

Description:

SUDAAN generates a PROGRAMMER ERROR when running RLOGIST and an invalid value is assigned to the PSULEV option in the NEST statement. SUDAAN appropriately issues a warning message about the invalid value for the PSULEV option, but, instead of halting, the procedure continues until it eventually generates the PROGRAMMER ERROR.

Work-around:

Entering a valid value in the PSULEV option of the NEST statement will prevent this PROGRAMMER ERROR.

Example:

The NEST statement in this example includes 3 variables, so valid values for the PSULEV option are 1, 2, or 3. The following procedure generates a PROGRAMMER ERROR, because 9 is an invalid value for the PSULEV option:

```
proc rlogist data=mydata design=wr;
nest survyear stratum psu / psulev=9 missunit;
weight weight;
model var1 = var2;
print ;
```

After changing PSULEV to 3, SUDAAN will not generate a PROGRAMMER ERROR:

```
proc rlogist data=mydata design=wr;
nest survyear stratum psu / psulev=3 missunit;
weight weight;
model var1 = var2;
print ;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.1
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

318. ALL Procedures: A PROGRAMMER ERROR may occur when you include a SUBPOPN statement in your program which defines a subpopulation of size 0.

Description:

A PROGRAMMER ERROR or other fault is generated when there is a SUBPOPN statement defining a subpopulation with no valid records. Instead of printing an error message and halting, SUDAAN continues with the calculations and eventually a PROGRAMMER ERROR is generated.

Work-around:

To avoid this error, you should confirm that the subpopulation defined by the SUBPOPN statement includes at least one valid record before you begin your analysis. You can do this in PROC RECORDS by including your SUBPOPN statement along with the COUNTREC option on the PROC RECORDS statement. This will cause PROC RECORDS to count and display the number of observations in the subpopulation. For the modeling procedures, you need to augment the SUBPOPN statement to remove records on which one or more MODEL variables are missing.

Example:

Suppose you want to make sure that the subpopulation defined in the following call to PROC REGRESS has valid records.

```
PROC REGRESS DATA=mydata FILETYPE=SAS DESIGN=WR;
WEIGHT WT;
NEST STRATUM PSU;
SUBPOPN RACE=2;
CLASS GENDER;
MODEL BMI = AGE GENDER;
```

The following statements will cause SUDAAN to compute and print the number of valid records in the specified subpopulation. Note that the SUBPOPN statement includes checks that all variables in the MODEL statement are non-missing. In this case it is enough to verify they are all positive values.

```
PROC RECORDS DATA=mydata FILETYPE=SAS NOPRINT COUNTREC;
SUBPOPN RACE=2 && BMI > 0 && AGE > 0 && GENDER > 0;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

319. CROSSTAB Procedure: A PROGRAMMER ERROR occurs when SMCOUNT and SMCONF options are specified on the PROC CROSSTAB statement.

Description:

A PROGRAMMER ERROR is generated in the CROSSTAB procedure when the SMCOUNT and SMCONF options are specified on the PROC statement.

Work-around:

Adding ROWSPCI to the PRINT statement will remove the PROGRAMMER ERROR. Another work-around is to remove the PRINT statement, allowing SUDAAN to print the default set of keywords.

Example:

This example will result in a PROGRAMMER ERROR:

```
PROC CROSSTAB DATA=mydata DESIGN=WR SMCOUNT=30 SMCONF=1 CONF_LIM=95;
WEIGHT WT;
NEST STRATUM PSU;
SUBPOPN RACE=2;
CLASS GENDER;
TABLES GENDER;
PRINT NSUM ROWPER SEROW;
```

In the next example, ROWSPCI has been added to the PRINT statement. This example will run without a PROGRAMMER ERROR:

```
PROC CROSSTAB DATA=mydata DESIGN=WR SMCOUNT=30 SMCONF=1 CONF_LIM=95;
WEIGHT WT;
NEST STRATUM PSU;
SUBPOPN RACE=2;
CLASS GENDER;
TABLES GENDER;
PRINT NSUM ROWPER SEROW ROWSPCI;
```

In the final example, the PRINT statement has been omitted. This example will print the default set of keywords, including ROWSPCI, without A PROGRAMMER ERROR:

```
PROC CROSSTAB DATA=mydata DESIGN=WR SMCOUNT=30 SMCONF=1 CONF_LIM=95;
WEIGHT WT;
NEST STRATUM PSU;
SUBPOPN RACE=2;
CLASS GENDER;
TABLES GENDER;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.1
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

320. WTADJUST Procedure: statistics are computed using the next-to-last iteration for non-convergent models

Description:

For WTADJUST models that do not converge within the number of iterations specified by the MAXITER parameter, SUDAAN computes several statistics using results from the next-to-last iteration instead of the last iteration. The affected statistics include the variance of beta, the weight adjustment factor, and all statistics derived from either the variance or the adjustment factor. Other statistics, such as beta, are computed as expected using results from the last iteration.

Work-around:

In general, results from a model that has not converged should not be used. There are several possible solutions:

- Increase the value of the MAXITER parameter.
- Remove, recode, or change some of the independent variables in the MODEL statement.
- Use the INITPARM statement to specify initial values for the betas (note that the default values are 0.00).
- Alter the value of TOL or P_EPSILON.

Continue to alter the model or procedure statements until the model properly converges.

Example:

Suppose the model defined by the following WTADJUST procedure requires 15 iterations to converge. Because MAXITER=10, the model will not converge. One would expect the output statistics to be computed using the results from the 10th iteration. However, statistics produced from the keywords sebeta, lowbeta, upbeta, adjfactor, and wtfinal are computed using the results from the 9th iteration.

```
proc wtadjust data="mydata"
              design=wr
              adjust=nonresponse
              maxiter=10;
    nest stratum psu;
    weight wt;
    model respondent = x y z;
    print beta sebeta lowbeta upbeta;
    idvar respondent;
    output idvar weight adjfactor wtfinal / filename="myout" replace;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

321. LOGISTIC (RLOGIST), MULTILOG, LOGLINK, and REGRESS Procedures: When R=EXCHANGEABLE is specified, there may a divide by 0 error if the value of the exchangeable correlation RHO is exactly equal to -1 or 1.

Description:

In rare cases the value of RHO may take on the values of 1 or -1, which can lead to a divide by 0 error in the calculation of the coefficient LAMBDA used in the formula for the exchangeable step. This error then leads to an abnormal exit from SUDAAN. Instead, SUDAAN should bound RHO away from ±1 and issue a warning.

Work-around:

If you suspect that the computed value of RHO for your data is equal or very nearly equal to ±1, use the INITRHO statement to specify a safe value of RHO for the exchangeable step.

Example:

In the following example, the INITRHO statement explicitly sets the value of RHO to use in the exchangeable step.

```
PROC REGRESS DATA=baddata R=EXCHANGEABLE;
NEST STRATUM PSU;
WEIGHT WT;

MODEL Y = A B X;
INITRHO -.999;

PRINT / BETAS=DEFAULT;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

322. LOGISTIC (RLOGIST), LOGLINK, MULTILOG, and REGRESS Procedures: SUDAAN hangs when there is only 1 record in each cluster (PSU variable on NEST Statement) and the R=Exchangeable option is specified.

Description:

When the maximum cluster size is 1 and the R=EXCHANGEABLE option is specified in the REGRESS, LOGISTIC, LOGLINK and MULTILOG procedures, SUDAAN correctly reports the data error, but then either hangs when executed from the SAS interactive environment or hangs without reporting the error when executed from the command prompt.

Work-around:

You can avoid this problem by checking the maximum size of the clusters in PROC CROSSTAB. To do this, use the same design statements in CROSSTAB as you have used in the modeling procedure, place the cluster variable on the CLASS statement, and print NSUM, or examine the frequency table provided by SUDAAN. If the largest sum is 1, then you should not use the EXCHANGEABLE correlation assumption for these data in the modeling procedures.

Example:

The following PROC CROSSTAB illustrates the work-around:

```
PROC CROSSTAB DATA=BADDATA NOMARG;
NEST _ONE_ CLUSTER;
WEIGHT _ONE_;
CLASS CLUSTER;
PRINT NSUM;
```

The results are shown here for a simple example:

```
Frequencies and Values for CLASS Variables
by: CLUSTER.

----------------------------------
CLUSTER         Frequency    Value
----------------------------------
Ordered
  Position:
  1                     1      102
Ordered
  Position:
  2                     1      112
Ordered
  Position:
  3                     1      141
Ordered
  Position:
  4                     1      142
Ordered
  Position:
  5                     1      152
Ordered
  Position:
```

```
  6                     1       181
Ordered
  Position:
  7                     1       182
Ordered
  Position:
  8                     1       211
---------------------------------
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

323. ALL Procedures: SUDAAN cannot write compressed datasets with uncompressed sizes larger than 4 GB

Description:

When SUDAAN tries to write a compressed SUDAAN dataset with an uncompressed size larger than 4 gigabytes, it produces a PROGRAMMER ERROR with the message "Pointer error in encode algorithm - unable to continue".

SUDAAN creates compressed datasets under two conditions, both of which are affected by this bug:

- The user requests a file of type SUDAAN or SUDXPORT on the OUTPUT statement without also providing the NOCOMP option.
- The user specifies the NOTSORTED option on a PROC statement, and either does not specify the OUTDATA option or specifies a SAS output file with the OUTDATA option. In this case, SUDAAN produces internal copies of the input datasets, which it then sorts and compresses.

Work-around:

When using the OUTPUT statement, provide the NOCOMP option to prevent SUDAAN from compressing the dataset. Another alternative is to choose an output file type other than SUDAAN or SUDXPORT.

For the NOTSORTED case, sort the dataset prior to running the procedure so that you can omit the NOTSORTED option from the procedure. If you sort using PROC RECORDS and create a sorted file of type SUDAAN or SUDXPORT, use the NOCOMP option on the OUTPUT statement to prevent SUDAAN from compressing the sorted dataset.

Examples:

1) Suppose you have a very large SAS dataset named BIGSAS. Attempting to convert it to the SUDXPORT format using PROC RECORDS will produce a PROGRAMMER ERROR if the dataset is large enough:

```
proc records data=bigsas filetype=sas noprint;
    output data="bigsdx" filetype=sudxport replace;
```

2) If BIGSAS is unsorted and you attempt to fit a model to the data, SUDAAN will produce a PROGRAMMER ERROR if the internal sorted version of the dataset is large enough:

```
proc rlogist data=bigsas filetype=sas design=wr notsorted;
    nest stratum psu;
    weight wt;
    class x1-x10000;
    model y = x1-x10000;
    print / betas=all;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

324. KAPMEIER Procedure: Incorrect values of STRHAZ variables in output dataset when more than 2 variables appear on STRHAZ statement

Description:

When the STRHAZ statement is included in the KAPMEIER procedure and the KMSTHZ keyword is either explicitly requested or implied on the output dataset, SUDAAN should copy the values of the STRHAZ variables onto each record of the output dataset. However, when more than 2 variables appear on the STRHAZ statement, SUDAAN copies the values incorrectly, resulting in incorrect values for some of the STRHAZ variables for some of the records. Other keywords on the output dataset, such as KM or SEKM, aren't affected by this bug.

Work-around:

Create a new variable whose value on each record is a combination of the values of the variables on the original STRHAZ statement, and use this new single variable in the PROC KAPMEIER call. The new variable should be created outside of SUDAAN before the PROC KAPMEIER call.

Example:

Suppose the following statements are included in a PROC KAPMEIER call, where each of the three CLASS variables has two levels.

```
class  papexam mamexam insurance;
strhaz papexam mamexam insurance;
output / kapmeier=all filename=km_out replace;
```

The OUTPUT keyword KMSTHZ is implied on the OUTPUT statement with KAPMEIER=ALL. Accordingly, the output dataset KM_OUT will include the STRHAZ variables PAPEXAM, MAMEXAM, and INSURANCE, but their values will be incorrect on some records. To work around this bug, first add a new variable, let's call it PMI, to the dataset before analyzing it with SUDAAN. PMI should have eight levels (2*2*2 = 8) with values assigned as follows:

PAPEXAM	MAMEXAM	INSURANCE	PMI
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Then, replace the original PROC KAPMEIER statements with these:

```
class  pmi;
strhaz pmi;
output / kapmeier=all filename=km_out replace;
```

System	Release Reported	Release Fixed
Windows	10.0.0	11.0.0
Solaris	10.0.0	11.0.0
Linux	10.0.0	11.0.0

325. LOGISTIC (RLOGIST), LOGLINK, MULTILOG, REGRESS and SURVIVAL Procedures: Incorrect ADJRR results with multiple terms on PREDMARG or CONDMARG if at least one term is an interaction involving a continuous variable

Description:

When multiple model terms are specified on a PREDMARG or CONDMARG statement with the ADJRR option, and at least one of those terms is an interaction involving a continuous variable, the adjusted risk ratios, associated standard errors and confidence intervals may be incorrect. If the terms are instead specified separately on their own PREDMARG or CONDMARG statements, the results are correct.

Work-around:

To compute adjusted risk ratios for multiple model terms, place each term on its own PREDMARG or CONDMARG statement instead of specifying them all on a single PREDMARG or CONDMARG statement.

Examples:

In the following example, a PREDMARG statement with the ADJRR option is specified. The statement includes two terms, each of which is a cross between a categorical and a continuous variable. As a result of the bug described here, some of the adjusted risk ratio statistics produced by SUDAAN will be incorrect.

```
proc rlogist data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class educ race;
    reflev educ=3 race=1;
    model y = race educ income educ*income race*income;
    predmarg race*income educ*income / income=(1,20) adjrr;
    print / pred_mrg=all predrisk=all;
```

To avoid the bug, place each term on its own PREDMARG statement, as in the following example:

```
proc rlogist data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class educ race;
    reflev educ=3 race=1;
    model y = race educ income educ*income race*income;
    predmarg race*income / income=(1,20) adjrr;
    predmarg educ*income / income=(1,20) adjrr;
    print / pred_mrg=all predrisk=all;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

326. SURVIVAL Procedure: Computing variances for conditional and predicted marginals may require a very long run time.

Description:

The SURVIVAL procedure may run for a very long time when computing variances for conditional and predicted marginals. In addition to cases when variance and/or standard error keywords are explicitly requested on the PRINT or OUTPUT statement, computing variances is also required when confidence intervals or test statistics are requested.

The actual run time is dependent on numerous factors, including the number of observations in the analysis population, the complexity of the model statement, the choice of sampling design, and the specifications of the computer running the analysis. Therefore, run times may vary widely from one analysis to another. However, we've included the following to illustrate just how long "a very long time" might be: We set up a SURVIVAL analysis with about 75,000 observations, a with-replacement sampling design, a single variable on the MODEL statement, and a single variable on the CONDMARG statement. The analysis required 5 seconds to run when the conditional marginal variance wasn't requested. After adding SECNDMRG to the PRINT statement, the analysis required approximately 50 hours to run.

Work-around:

Specifying the TIES=BRESLOW option on the MODEL statement may make the procedure run faster than TIES=EFRON (the default), although it still may take a long time. In the example described above, the procedure finished in 5 hours after switching to TIES=BRESLOW.

Example:

The following SURVIVAL example requests the confidence interval for the conditional marginal, which triggers the computation of the conditional marginal variance. Therefore, this analysis may take a long time to run.

```
proc survival data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class spd2;
    event mortstat;
    model interval = spd2;
    condmarg spd2;
    print beta condmrg lowcm upcm;
run;
```

Adding TIES=BRESLOW to the MODEL statement may help it run faster:

```
    model interval = spd2 / ties=breslow;
```

System	Release Reported	Release Fixed
Windows	10.0.1	not fixed
Solaris	10.0.1	not fixed
Linux	10.0.1	not fixed

327. ALL Procedures: AND is not given precedence over OR in SUBPOPN statements

Description:

If a SUBPOPN statement includes three or more conditions connected with AND and OR operators, and parentheses haven't been provided to explicitly group the conditions, SUDAAN evaluates them from left to right instead of following the standard practice of evaluating the AND operators before the OR operators.

Work-around:

Use parentheses to group the conditions in a SUBPOPN expression whenever multiple conditions are connected with AND or OR operators.

Examples:

Suppose an analysis should include all records from 2009 in addition to records from January of 2010. One would expect a SUBPOPN statement like the one in the following example to produce the desired set of records:

```
proc rlogist data=mydata design=wr;
    subpopn year=2009 or year=2010 and month=1;
    nest stratum psu;
    weight wt;
    class educ race;
    model y = race educ income educ*income race*income;
```

However, SUDAAN interprets this SUBPOPN statement as if it had been written as:

```
    subpopn (year=2009 or year=2010) and month=1;
```

As a result, only records from January of 2009 and January of 2010 will be included in the analysis. To avoid the bug, use parentheses to explicitly group the conditions:

```
    subpopn year=2009 or (year=2010 and month=1);
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

328. CROSSTAB, DESCRIPT, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, RATIO, and REGRESS Procedures: PROGRAMMER ERROR when MI data is specified using the MI_VAR statement and DESIGN=SRS is specified on the PROC statement.

Description:

If MI data is specified using the MI_VAR statement and DESIGN=SRS is specified on the PROC statement, SUDAAN produces a PROGRAMMER ERROR.

Work-around:

One workaround is to change the DESIGN option to DESIGN=STRWR and set the NEST and WEIGHT statement variables to _ONE_. This will produce the same results as DESIGN=SRS.

Another workaround is to avoid using the MI_VAR statement. Instead of storing the multiply imputed variables in a single dataset and specifying them with MI_VAR, another option is to store them in separate datasets and identify them via either the MI_COUNT option on the PROC statement or via the MI_FILES statement (see *Chapter 5* of the SUDAAN 11 User's Manual for examples).

Examples:

The following job will produce a PROGRAMMER ERROR:

```
proc crosstab data=mydata design=srs;
    mi_var race1 race2 race3;
    class race1;
    table race1;
    print nsum;
    run;
```

To implement the DESIGN=STRWR workaround, use this instead:

```
proc crosstab data=mydata design=strwr;
    nest _one_;
    weight _one_;
    mi_var race1 race2 race3;
    class race1;
    table race1;
    print nsum;
    run;
```

To implement the MI_COUNT workaround, create three datasets named MYDATA1, MYDATA2, and MYDATA3. Create a variable named RACE on each one, setting it equal to RACE1 on MYDATA1, RACE2 on MYDATA2, and RACE3 on MYDATA3. All other variables should be identical on the three datasets. Then submit the following job:

```
proc crosstab data=mydata1 design=srs mi_count=3;
    class race;
    table race;
    print nsum;
    run;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

329. ALL: SUDAAN.ENV file is ignored when placed in the same directory as the SUDAAN executable file

Description:

For command-line SUDAAN, the user should be able to configure default PRINT and OUTPUT environment settings by placing them in a file named SUDAAN.ENV and placing that file in the same directory as the SUDAAN executable file. However, SUDAAN ignores the settings in SUDAAN.ENV when it is located in the same directory as the SUDAAN executable file.

Work-around:

Command-line SUDAAN does correctly read the SUDAAN.ENV settings when that file is placed in the directory from which you execute SUDAAN. Therefore, instead of placing SUDAAN.ENV in the same directory as the SUDAAN executable file, you can place it in the directory from which you execute SUDAAN. If you execute SUDAAN from multiple directories, you will need a copy of SUDAAN.ENV in each one.

Example:

Suppose your command-line SUDAAN executable file has been installed in the directory:

C:\Program Files\SUDAAN\Release10.0.1\DOS

Suppose you have stored your SUDAAN programs in the directory:

C:\Projects\Project12

and this is the directory from which you execute SUDAAN. If you store your SUDAAN.ENV file in "C:\Program Files\SUDAAN\Release10.0.1\DOS", the settings defined within it will be ignored. However, if you store it in "C:\Projects\Project12", the settings will be applied.

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

330. SURVIVAL Procedure: Number of iterations computed or reported by SURVIVAL is off by 1, sometimes leading to non-convergent models

Description:

The iterative model-fitting procedure in SURVIVAL should continue until the model converges or the number of iterations reaches the limit specified by the MAXITER parameter, whichever occurs first. Suppose MAXITER=*n* and the model converges in *k* iterations. The bug manifests itself slightly differently depending on the relationship of *k* to *n*.

When k = n, SURVIVAL should report that the model converged in *n* iterations. However, it reports that the model did not converge in *n*-1 iterations.

When *k* > *n*, SURVIVAL should report that the model did not converge in *n* iterations. However, it reports that the model did not converge in *n*-1 iterations.

When *k* < *n,* SURVIVAL should report that the model converged in *k* iterations. However, it reports that the model converged in *k*+1 iterations.

Work-around:

To work around the *k* = *n* and *k* > *n* cases, set the MAXITER parameter to a value that is 1 higher than the actual number of iterations you wish SURVIVAL to compute.

The *k* < *n* case has no work-around. Note that in this case, the issue is not the non-convergence but the number of iterations needed for convergence.

Example:

Suppose your SURVIVAL model converges in 6 iterations and your PROC statement looks like:

```
proc survival data=mydata design=wr maxiter=6;
    ... ;
```

SURVIVAL should report that the model has converged in 6 iterations. However, it prints the following message in the output file:

```
Warning: SURVIVAL has not converged to a solution in 5 iterations.
```

Setting MAXITER to 7 or higher would allow the model to converge, and the message printed in the output file would be:

```
SURVIVAL has converged to a solution in 7 iterations.
```

Note that the reported number of iterations is 7 even though the model actually converged in 6 iterations.

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

331. SURVIVAL Procedure: Incorrect estimates or PROGRAMMER ERROR in discrete models when the starting interval is greater than the total number of intervals

Description:

For discrete proportional hazards models, the SURVIVAL procedure estimates *m* baseline hazards, where *m* is defined by the INTERVALS or LAMBDAS option on the MODEL statement. The MODEL statement allows the user to specify two dependent variables, *t1* and *t2*, where *t1* represents the starting interval when the individual enters the study, and *t2* represents the interval in which the individual undergoes an event or is censored.

SUDAAN should ignore records on the input dataset where *t1* > *m*. These records correspond to individuals who entered the study outside the time period of interest and are not a part of the study population. However, SUDAAN does not handle these records correctly. As a result, SUDAAN produces incorrect estimates for the betas, variances, and other statistics. In cases where *t1* is very large, SUDAAN may issue a PROGRAMMER ERROR and terminate before producing any results.

Work-around:

Before running the analysis, delete all observations from the dataset where *t1* > *m*.

Note that this work-around contradicts the standard advice when using Taylor series designs, which is to use the SUBPOPN or SUBPOPX statement instead of deleting observations from the dataset. Deleting observations from the dataset may affect the counts of PSUs within the strata, which in turn may affect the variance estimates. However, in the situation affected by this bug, records where the start time is greater than the number of intervals are not actually a part of the study population, and therefore they should not contribute to the PSU counts.

Example:

The following example fits a discrete proportional hazards model with 10 intervals, and the starting interval is specified on the MODEL statement as the dependent variable *t1*:

```
proc survival data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class agegrp gender;
    event censor_flag;
    model t1 t2 = agegrp gender / intervals=10;
    print beta sebeta;
    run;
```

Suppose the mydata dataset includes records with *t1* > 10, for example *t1*=11, *t1*=12, ..., *t1*=20. Then the values computed for beta and sebeta will be incorrect. If the values for *t1* are very large, for example *t1*=500, then SUDAAN may instead produce a PROGRAMMER ERROR.

SURVIVAL should produce correct results once the records with *t1* > 10 are removed from the dataset. One problem you may encounter is that after removing records, you may end up with PSUs containing just 1 record or strata containing just 1 PSU. If this is the case, then you should add the MISSUNIT option to the NEST statement:

```
data mydata_reduced;
    set mydata;
    if t1 <= 10 then output;
    run;

proc survival data=mydata_reduced design=wr;
    nest stratum psu / missunit;
    weight wt;
    class agegrp gender;
    event censor_flag;
    model t1 t2 = agegrp gender / intervals=10;
    print beta sebeta;
    run;
```

System	Release Reported	Release Fixed
Windows	10.0.0	11.0.1
Solaris	10.0.0	11.0.1
Linux	10.0.0	11.0.1

356. ALL Procedures: "LINESIZE too small" error when CLASS variables have long formatted values

Description:

When the CLASS statement is present, SUDAAN prints a table displaying each CLASS variable and the levels associated with the variable. When formats have been associated with CLASS variables, the CLASS table additionally displays the formatted values for each variable level. If the formatted value is so long that a row of the CLASS table won't fit on a single line, SUDAAN issues a LIMITS ERROR and halts. The error message indicates that the value of the LINESIZE option is too small.

Although this bug affects both SAS-callable and standalone SUDAAN, it is much less likely to be encountered in standalone. SAS datasets allow longer formatted values than the file types supported in standalone SUDAAN. The shorter formatted values allowed in standalone should fit on a single line unless the LINESIZE option has been set to a very small value (around 50 or lower).

Work-around:

This bug has three possible work-arounds:

1. Increase the value of the LINESIZE option. Note that the maximum LINESIZE supported by SUDAAN is 255, so if your formatted values have more than 255 characters, then this work-around is not suitable. Also note that you should not use SUDAAN's SETENV statement to change LINESIZE. In SAS-callable SUDAAN, you should use SAS's OPTIONS statement. In standalone SUDAAN, you should set the LINESIZE in your SUDAAN.ENV file.
2. Suppress the table displaying formatted values of the CLASS variables by adding the NOFREQ option to the CLASS statement.
3. Re-format your variables to shorten the formatted values.

Example:

In the following SAS-callable program, level 1 of the PROGRAM variable is formatted with a value whose length is 90 characters. With a LINESIZE of 100, the CLASS table row for this level won't fit on a single line. (The row includes more than just the formatted value, so the length of the row is actually about 108 characters.)

```
options linesize=100;

proc format;
    value progf
        1 = "123456789 123456789 123456789 123456789 123456789 123456789
123456789 123456789 1234567890"
        2 = "abcd"
        ;
    run;

proc crosstab data=mydata design=srs;
    class program;
    table program;
    print nsum / style=nchs;
    rformat program progf.;
    run;
```

Running this program produces a LIMITS ERROR in the log file. To work around the error, you could increase the value of the LINESIZE option:

```
options linesize=110;
```

Or suppress the CLASS variable table using the NOFREQ option:

```
class program / nofreq;
```

Or shorten the formatted value:

```
value progf
        1 = "123456789 123456789 123456789 1234567890"
        2 = "abcd"
        ;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0

360. CROSSTAB, DESCRIPT, LOGISTIC (RLOGIST), RATIO, VARGEN, WTADJUST, and WTADJX Procedures: PROGRAMMER ERROR when printing large tables with STYLE=NCHS.

Description:

When printing results using the STYLE=NCHS option on the PRINT statement, SUDAAN produces a PROGRAMMER ERROR if the table containing the results has more than approximately 3,000 rows. The STYLE option is available only for procedures that support the TABLES statement.

Work-around:

To work around this bug, you must reduce the number of rows in the printed table. This can be accomplished in several ways:

- Use the NDIMROW option on the PRINT statement to reduce the number of variables that are nested when forming the rows of the table. If you've already included the NDIMROW option, try setting it to a lower value. If you haven't already included the NDIMROW option, try adding NDIMROW=1.
- Use the default STYLE=BOX option instead of STYLE=NCHS on the PRINT statement.
- Avoid printing the large table altogether by adding the NOPRINT option to the PROC statement and removing the PRINT statement. Instead of printing the table, output the results to a data file using the OUTPUT statement.

Example:

In the following example, the table containing the requested statistics has 3 dimensions. The first dimension represents the levels of the COUNTY variable and has a size of 1,000. The second dimension represents the levels of the EDUCATION variable and has a size of 5. The third dimension represents the requested statistics and has a size of 4. The STYLE=NCHS option instructs SUDAAN to create the table rows by nesting the first 2 dimensions, producing a table with 5,000 rows and 4 columns. SUDAAN produces a PROGRAMMER ERROR because the number of rows exceeds 3,000.

```
proc crosstab data=mydata design=wr;
    nest stratum psu;
    weight wt;
    subgroup county education;
    levels     1000         5;
    tables county*education;
    print nsum wsum rowper serow / style=nchs;
run;
```

Specifying NDIMROW=1 instructs SUDAAN to nest just 1 dimension in the table rows, which is equivalent to no nesting. It chooses the second dimension for the table rows, and it uses the first dimension to create separate tables. It uses the third dimension for the table columns. As a result, SUDAAN produces 1,000 tables, each with 5 rows and 4 columns.

```
    print nsum wsum rowper serow / style=nchs ndimrow=1;
```

With the default STYLE=BOX, SUDAAN uses the first dimension as the table rows and the second dimension as the table columns. It prints the third dimension inside each table cell. As a result, SUDAAN produces a table with 1,000 rows and 5 columns.

```
print nsum wsum rowper serow / style=box;
```

Using the NOPRINT option and the OUTPUT statement would avoid the problem by simply not printing the table. Viewing the results would then require examining or manipulating the output dataset in a later step.

```
proc crosstab data=mydata design=wr noprint;
    nest stratum psu;
    weight wt;
    subgroup county education;
    levels     1000         5;
    tables county*education;
    output nsum wsum rowper serow / filename=outdata replace;
run;
```

System	Release Reported	Release Fixed
Windows	11.0.0	11.0.1
Solaris	11.0.0	11.0.1
Linux	11.0.0	11.0.1

362. All Procedures: PROGRAMMER ERROR when writing to a SASXPORT file if the PSUDATA option is used

Description:

SUDAAN produces a PROGRAMMER ERROR if the PSUDATA option is used on the PROC statement and the OUTPUT statement specifies an output file with filetype=SASXPORT. This bug affects standalone SUDAAN but not SAS-callable SUDAAN, because SAS-callable SUDAAN does not support the SASXPORT filetype.

Work-around:

Instead of writing the output to a SASXPORT file directly, first write to a SUDXPORT file, and then convert the SUDXPORT file to a SASXPORT file using the RECORDS procedure.

Example:

The following SUDAAN procedure will produce a PROGRAMMER ERROR:

```
proc crosstab data="c:\myproject\mydata.sdx"
              psudata="c:\myproject\mydata_psu.sdx"
              filetype=sudxport
              design=wr
              noprint;
    nest stratum psu;
    weight wt;
    subgroup gender;
    levels 2;
    table gender;
    output nsum wsum totper setot
           / filename="c:\myproject\cross.xpt"
             filetype=sasxport
             replace;
```

To avoid the PROGRAMMER ERROR, specify a SUDXPORT file on the OUTPUT statement and then convert the SUDXPORT file to SASXPORT using PROC RECORDS:

```
proc crosstab data="c:\myproject\mydata.sdx"
              psudata="c:\myproject\mydata_psu.sdx"
              filetype=sudxport
              design=wr
              noprint;
    nest stratum psu;
    weight wt;
    subgroup gender;
    levels 2;
    table gender;
    output nsum wsum totper setot
           / filename="c:\myproject\cross.sdx"
             filetype=sudxport
             replace;

proc records data="c:\myproject\cross.sdx"
             filetype=sudxport
             noprint;
```

```
output / filename="c:\myproject\cross.xpt"
         filetype=sasxport
         replace;
```

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.1
Solaris	10.0.1	11.0.1
Linux	10.0.1	11.0.1

367. SURVIVAL Procedure: Illegal variable name error when LAMBDA or SELAMBDA is output to a SAS dataset

Description:

If the LAMBDA or SELAMBDA keyword appears on the OUTPUT statement in the SURVIVAL procedure, and the output dataset has FILETYPE=SAS, then SAS will issue the error message "The variable name is illegal", and the output dataset won't be created. Specifying the BASERATE group (the group containing the LAMBDA keywords) on the OUTPUT statement produces the same behavior.

Work-around:

The bug is caused by SUDAAN attempting to add a variable with a blank name to the output dataset, and variables with blank names aren't allowed on SAS datasets. However, variables with blank names are allowed on SUDXPORT datasets. To work around the bug, first create an output dataset in SURVIVAL with FILETYPE=SUDXPORT instead of FILETYPE=SAS. Then use the RECORDS procedure to convert the SUDXPORT dataset to a SAS dataset, being sure to keep only the variables of interest.

The variable with the blank name is STRHAZ. As a result, the STRHAZ variable will not be available when converting the SUDXPORT dataset to a SAS dataset.

Example:

The following SURVIVAL procedure attempts to output the LAMBDA and SELAMBDA keywords to a SAS dataset named LAMBDAS:

```
proc survival data=mydata design=wr noprint;
    nest stratum psu;
    weight wt;
    event mortstat;
    model t1 = income anyaid / intervals=10;;
    output lambda selambda / filename=lambdas filetype=sas replace;
    run;
```

An error will be printed to the SAS log file, and the LAMBDAS dataset won't be created. To work around the bug, first replace the OUTPUT statement with one that creates a SUDXPORT dataset:

```
    output lambda selambda / filename="c:\lambdas.sdx"
                             filetype=sudxport replace;
```

Then use the RECORDS procedure to convert the SUDXPORT dataset to a SAS dataset:

```
proc records data="c:\lambdas.sdx" filetype=sudxport replace;
    output interval lambda selambda / filename=lambdas
                                      filetype=sas replace;
    run;
```

On the OUTPUT statement, it is important to explicitly specify the variables that should be saved to the SAS dataset. In this case, the INTERVAL, LAMBDA, and SELAMBDA variables are specified. TABLENO is another variable you may want to specify. If no variables are specified,

SUDAAN will attempt to add all the variables to the SAS dataset, including the blank-named variable, which will again produce the error.

System	Release Reported	Release Fixed
Windows	10.0.1	11.0.0
Solaris	10.0.1	11.0.0
Linux	10.0.1	11.0.0