Bugs Affecting SUDAAN 11.0.0

This list was updated on July 15, 2013. The most recent list can be found at http://sudaansupport.rti.org/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.

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

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

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

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

332. ALL Procedures: Formats are not applied to PSUDATA variables when the NOTSORTED option is used

333. DESCRIPT, LOGISTIC (RLOGIST), WTADJUST and WTADJX Procedures: SEMANTIC ERROR when DDF is on an OUTPUT statement with certain other keywords

334. IMPUTE Procedure: IMPBY variables are not formatted correctly in PRINT tables

335. MULTILOG Procedure: Extraneous row in ITBETAS output table with exchangeable working correlations

336. All procedures except RECORDS: PROGRAMMER ERROR when opening SPSS data files

337. ALL Procedures: SUDAAN produces a DATA ERROR when reading an ASCII dataset it just created.

338. ALL Procedures: Unable to access variables with long variable names saved in SASXPORT files created by SUDAAN

339. CROSSTAB, DESCRIPT, KAPMEIER, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, RATIO, REGRESS, SURVIVAL, VARGEN, WTADJUST, and WTADJX Procedures: Using the REPDATA parameter causes a PROGRAMMER ERROR.

340. ALL Procedures: In Standalone SUDAAN formats defined in a LEVFILE are not assigned to the variables if the FORMAT statement is used.

341. All Procedures: Using the NEWVAR statement to create a variable that already exists on the input file in combination with the NOTSORTED option on the PROC statement causes a PROGRAMMER ERROR.

342. LOGLINK and REGRESS Procedures: DATA ERROR when the weighted sum of the dependent variable is negative or zero

343. All Procedures: An error generated by one value of the variables listed in the BY statement prevents SUDAAN from producing results for the remaining values of the BY variables.

344. KAPMEIER, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, SURVIVAL, WTADJUST, and WTADJX Procedures: System error when computing some variance-derived statistics when only 1 PSU in a sampling stratum

345. All Procedures: Syntax error generated when using minus signs on NEWVAR, SUBPOPX, PARAMETER, and X statements

346. CROSSTAB, DESCRIPT, LOGISTIC (RLOGIST), RATIO, VARGEN, WTADJUST, and WTADJX Procedures: SYNTAX ERROR when variables are specified using dash notation on TABLE statement

347. WTADJUST and WTADJX Procedures: Incorrect values for the CNTLTOTAL, DIFFWT, and MARGADJ keywords when the REFLEVEL statement is used

348. All Procedures: Access violation when REPLACE or APPEND not specified for RTF print format.

349. LOGLINK, MULTILOG and REGRESS Procedures: Unable to output keywords from the VARIANCE group

350. LOGISTIC (RLOGIST), WTADJUST and WTADJX Procedures: Unable to output the default keywords from the MEANCOV, PCTCOV, RHATCOV, and TOTCOV groups

351. LOGISTIC (RLOGIST), LOGLINK, MULTILOG, REGRESS, WTADJUST and WTADJX Procedures: Fatal memory allocation error

352. ALL Procedures: PROGRAMMER ERROR when a variable on the RFORMAT statement does not appear on the dataset

353. VARGEN Procedure: A PROGRAMMER ERROR is encountered when using PROC statement option DESIGN=SRS.

354. LOGLINK Procedure: Floating point error when the dependent variable has the same value on every record.

355. LOGLINK, MULTILOG, and REGRESS Procedures: Unable to print the DDF keyword

357. All Procedures: FILE ERROR when the definition of a NEWVAR variable includes a previous NEWVAR variable.

358. VARGEN Procedure: Using an expression containing nested parentheses may cause a PROGRAMMER ERROR.

359. All Procedures: Floating point error when class variable contains large numbers for SAS-Callable SUDAAN.

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

361. LOGISTIC (RLOGIST) and LOGLINK Procedures: Additional iteration in convergence algorithm for exchangeable working correlations

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

363. VARGEN Procedure: PROGRAMMER ERROR when contrasts and multiple tables are requested for BRR designs

364. VARGEN Procedure: Incorrect standard errors for subpopulation estimates with replicate designs

365. VARGEN Procedure: Incorrect values for VARIABLE on output dataset when contrasts and multiple tables are requested

366. All Procedures: Unexpected error messages when character variables appear on certain statements

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

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

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

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

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

332. ALL Procedures: Formats are not applied to PSUDATA variables when the NOTSORTED option is used

Description:

If the NOTSORTED option appears on the PROC statement, and a variable from the PSUDATA file appears on the RFORMAT statement, then SUDAAN displays the unformatted values of that variable in the printed results instead of the formatted values.

Work-around:

Working around this bug is possible using two different approaches.

The first approach is to omit the NOTSORTED option. Choosing this method requires you to sort both the DATA file and the PSUDATA file by the NEST variables prior to running the procedure.

The second approach is to make sure any variable that should be formatted appears on the DATA file. Prior to running the procedure, you should move such variables from the PSUDATA file to the DATA file.

Example:

Suppose the variable named REGION appears on your PSUDATA file, and you want to format its values in the printed results. Also suppose that your DATA file isn't already sorted by the NEST variables. You might do something like this:

```
proc format;
    value regf
        1 = "east"
        2 = "west"
        3 = "south"
        ;
    run;

proc crosstab data=mydata psudata=mypsudata notsorted design=wr;
    nest stratum psu;
    weight wt;
    class region gender;
    table region * gender;
    rformat region regf.;
    print nsum rowper serow;
    run;
```

In this case, SUDAAN will display the values of the REGION variable as "1", "2", and "3" in the CROSSTAB table. If you sort the DATA file ahead of time and omit the NOTSORTED option, then SUDAAN will correctly display the values of REGION as "east", "west", and "south".

System	Release Reported	Release Fixed
Windows	11.0.0	not fixed
Solaris	11.0.0	not fixed
Linux	11.0.0	not fixed

333. DESCRIPT, LOGISTIC (RLOGIST), WTADJUST and WTADJX Procedures: SEMANTIC ERROR when DDF is on an OUTPUT statement with certain other keywords

Description:

When the DDF keyword and another specific keyword are listed together on a single OUTPUT statement in PROC DESCRIPT, RLOGIST, WTADJUST and WTADJX, SUDAAN produces the following semantic error: "On OUTPUT statement #1 please specify statistics from exactly one output group using the option group_name=DEFAULT or the option group_name=ALL". SUDAAN produces this semantic error even when DDF and the other keyword belong to the same output group. The affected keyword combinations are:

In DESCRIPT:
OUTPUT DDF TOTAL

In RLOGIST, WTADJUST, and WTADJX:
OUTPUT DDF MEAN
OUTPUT DDF PERCENT
OUTPUT DDF RHAT (but only when the VAR statement is also present)

Work-arounds:

One method for working around this bug is to specify the keywords on separate OUTPUT statements instead of on a single output statement. Using this method, the two keywords will be saved to separate output files.

Another method is to remove the explicit references to the keywords from the OUTPUT statement and instead request them using the group_name=ALL syntax. Using this method, the two keywords will be saved to a single output file, but additional keywords will be also computed and saved to the output file.

Example:

The following statements will produce the semantic error even though the DDF and MEAN keywords both belong to the MEANCOV output group in RLOGIST:

```
proc rlogist data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class region gender;
    model resp = region gender;
    var cig;
    output ddf mean / filename=out_meancov replace;
    run;
```

By placing the DDF and MEAN keywords on separate OUTPUT statements, the semantic error is avoided:

```
proc rlogist data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class region gender;
```

```
        model resp = region gender;
        var cig;
        output ddf / filename=out_ddf replace;
        output mean / filename=out_mean replace;
        run;
```

The semantic error is also avoided when requesting the entire MEANCOV group instead of requesting DDF and MEAN explicitly:

```
    proc rlogist data=mydata design=wr;
        nest stratum psu;
        weight wt;
        class region gender;
        model resp = region gender;
        var cig;
        output / meancov=all filename=out_meancov replace;
        run;
```

System	**Release Reported**	**Release Fixed**
Windows	11.0.0	not fixed
Solaris	11.0.0	not fixed
Linux	11.0.0	not fixed

334. IMPUTE Procedure: IMPBY variables are not formatted correctly in PRINT tables

Description:

The FORMAT (RFORMAT) statement does not apply to IMPBY variables in PRINT tables produced by the IMPUTE procedure. When the table is printed, the numeric level values are printed rather than the description provided by the format.

Work-around:

There is no work around for this problem.

Example:

The following example produces a printed table of donor statistics cross-classified by the variables racemom and educ. The RFORMAT statements specify descriptions for each level of these two variables. However, the table will show the numeric level values rather than the descriptions for these two variables.

```
proc impute data=wicwage notsorted;
    impby racemom educ;
    impvar babywgt;
    rformat racemom race.;
    rformat educ ed.;
    print / donorstat=default;

    rtitle "Imputing Baby Weight";
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

335. MULTILOG Procedure: Extraneous row in ITBETAS output table with exchangeable working correlations

Description:

When assuming exchangeable working correlations in the MULTILOG procedure (that is, when either the R=EXCHANGEABLE or RSTEPS=*n*>0 parameter is specified), SUDAAN sometimes inserts an extraneous set of betas (regression coefficient estimates) into the next-to-last row of the exchangeable portion of the ITBETAS output table. This set of betas also contributes to the reported number of iterations required for the exchangeable step to converge, making that number one greater than it should be. The extraneous row represents a set of betas that were rejected during the convergence algorithm for failing to improve the likelihood when compared to the previous iteration.

Work-around:

No work-around is available for this bug. The extraneous row in the table may be safely ignored.

Example:

Consider the following SUDAAN analysis:

```
proc multilog data=mydata design=wr r=exchangeable;
    nest stratum psu;
    weight wt;
    class group age gender;
    model group = age gender;
    print beta sebeta itbeta;
run;
```

If the convergence algorithm in one of the exchangeable steps (step > 0) rejects a set of betas, the ITBETAS table will include an extraneous row. For example, the betas for the first couple of terms of the model may look like this in the printed ITBETAS table:

```
for: Exchangeable Step = 1.

------------------------------------------------------------------
Iteration              Independent Variables and Effects
   GROUP (log-                                              AGE
     odds)                      Intercept                   1
------------------------------------------------------------------
0
   1 vs 4                    3.5098996735          1.4615414746
   2 vs 4                    3.2509864612          0.5918951071
   3 vs 4                    1.0088636412         -0.9158914912
1
   1 vs 4                    3.4365307157          1.6320917865
   2 vs 4                    3.1748479704          0.7655687843
   3 vs 4                    0.8561980635         -0.8061795300
2
   1 vs 4                    3.3631994196          1.4965533627
   2 vs 4                    3.0994801423          0.6275223137
   3 vs 4                    0.7631895882         -0.9405012131
3
   1 vs 4                    3.4365306807          1.6320917219
   2 vs 4                    3.1748479345          0.7655687184
   3 vs 4                      0.8561980192         -0.8061795941
--------------------------------------------------------------------
```

Looking at just the intercept term for the 1 vs 4 response, we see the following progression of betas through the iterations:

Iteration	BETA
0	3.5098996735
1	3.4365307157
2	3.3631994196
3	3.4365306807

The betas for iterations 1 and 3 are very similar to each other, but the beta for iteration 2 isn't similar to either of them, contrary to what one would expect from a convergent process. The explanation is that the beta for iteration 2 was actually ignored when determining convergence and should have been omitted from the table. The other betas in the ITBETAS table show a similar pattern.

System	Release Reported	Release Fixed
Windows	11.0.0	not fixed
Solaris	11.0.0	not fixed
Linux	11.0.0	not fixed

336. All procedures except RECORDS: PROGRAMMER ERROR when opening SPSS data files

Description:

SUDAAN issues a PROGRAMMER ERROR and halts whenever an SPSS data file is specified as the input file to a procedure other than PROC RECORDS.

This bug affects only the 32-bit Windows Standalone and Command Prompt platforms. SPSS data files are unsupported on other platforms.

Work-around:

Before running the analysis, convert the SPSS data file to SUDXPORT format using PROC RECORDS. Use the SUDXPORT file as the input file instead of the SPSS file.

Example:

The following procedure attempts to read an SPSS dataset and therefore will produce a PROGRAMMER ERROR:

```
proc crosstab data="c:\myproject\data\mydata.sav"
              filetype=spss
              design=wr;
    nest stratum psu;
    weight wt;
    class age gender;
    table age gender;
    print nsum totper setot;
```

To work around the bug, first convert the SPSS file to SUDXPORT format:

```
proc records data="c:\myproject\data\mydata.sav"
             filetype=spss
             noprint;
    output / filename="c:\myproject\data\mydata.sdx"
             filetype=sudxport;
```

Then change the original procedure to read the SUDXPORT file instead of the SPSS file:

```
proc crosstab data="c:\myproject\data\mydata.sdx"
              filetype=sudxport
              design=wr;
    nest stratum psu;
    weight wt;
    class age gender;
    table age gender;
    print nsum totper setot;
```

System	Release Reported	Release Fixed
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

337. ALL Procedures: SUDAAN produces a DATA ERROR when reading an ASCII dataset it just created.

Description:

SUDAAN cannot read an ASCII data file that it just created, specifically when the created data file contains categorical variables, none of which have labels associated with any of its levels. SUDAAN instead produces an error message.

Work-around:

There is no work-around for this bug.

Example:

In this example, *mydata.sdx* is an input data file containing categorical variables. None of the categorical variables have labels associated with its levels. Below, the first PROC RECORDS produces an ASCII data file from *mydata.sdx*. The second PROC RECORDS produces a DATA ERROR when it tries to read that ASCII data file.

```
proc records data="mydata.sdx" filetype=sudxport contents;
    print / maxrec = 10;
    output / filename="myasciidata" filetype=ascii replace;

proc records data="myasciidata" filetype=ascii contents countrec;
    print / maxrec=5;
```

System	**Release Reported**	**Release Fixed**
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

338. ALL Procedures: Unable to access variables with long variable names saved in SASXPORT files created by SUDAAN

Description:

Variable names on SASXPORT files are limited to 8 characters or fewer. When an output dataset, created in the OUTPUT statement with the option FILETYPE=SASXPORT, has variables with long variable names, SUDAAN truncates the variable names to the first 8 characters to meet the requirements of SASXPORT files. Using the NAMEFILE option in the same OUTPUT statement, the user has the option to create a SASXPORT file that will contain a mapping between the truncated names and the original names. SUDAAN can then use this mapping file (NAMEFILE) when reading the dataset. This makes the name truncation process somewhat transparent and allows the user to always refer to the variables by their original, long names.

SUDAAN does not create the NAMEFILE properly. As a result, when a user creates a SASXPORT file containing long variable names and later uses it as an input file, SUDAAN doesn't recognize the long variable names as valid variable names.

This does not affect SAS-Callable SUDAAN, only Standalone SUDAAN.

Work-around:

Using the RBY statement when creating the SASXPORT file forces SUDAAN to create the NAMEFILE properly. If the user does not want to perform the analysis by any particular variable, the user can use the keyword _ONE_ in the RBY statement, as shown below:
RBY _ONE_;.

Example:

In this example, suppose you want to save the variable PRED_LOWRR to a SASXPORT file. Because PRED_LOWRR has more than 8 characters, its full name cannot be saved in a SASXPORT file, so you also create a NAMEFILE for storing the mapping between the truncated name and the original name:

```
proc rlogist data="mydata.sdx" filetype=sudxport design=srs;
    class treatment;
    model pass = treatment age;
    predmarg race / adjrr;
    output pred_lowrr / filename="pred_lowrr.stx"
                        filetype=sasxport
                        replace
                        namefile="pred_lowrr_names.stx";
```

SUDAAN prints warnings to the output file to confirm that it truncated variable names:

```
FILE WARNING:
Truncating variable name 'PRED_LOWRR' to 'PRED_LOW' for SAS transport
output
```

When reading the SASXPORT dataset in SUDAAN Standalone, you should be able to specify the NAMEFILE and access the variable PRED_LOWRR by using the long variable name:

```
proc records data="pred_lowrr.stx"
```

```
                 filetype=sasxport
                 namefile="pred_lowrr_names.stx";
        print pred_lowrr;
```

However, SUDAAN does not recognize the long variable name. Instead, it prints a SYNTAX ERROR:

```
SYNTAX ERROR in Statement 9:
(Message 314)
Undefined or unavailable keyword name 'PRED_LOWRR'
```

This error may be avoided by inserting an RBY statement in the RLOGIST procedure that creates the SASXPORT output file and accompanying NAMEFILE:

```
proc rlogist data="mydata.sdx" filetype=sudxport design=srs;
    rby _one_;
    class treatment;
    model pass = treatment age;
    predmarg race / adjrr;
    output pred_lowrr / filename="pred_lowrr.stx"
                        filetype=sasxport
                        replace
                        namefile="pred_lowrr_names.stx";
```

System	**Release Reported**	**Release Fixed**
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

339. CROSSTAB, DESCRIPT, KAPMEIER, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, RATIO, REGRESS, SURVIVAL, VARGEN, WTADJUST, and WTADJX Procedures: Using the REPDATA parameter causes a PROGRAMMER ERROR.

Description:

Using the REPDATA parameter with any procedure that allows this parameter causes SUDAAN to terminate with a PROGRAMMER ERROR.

Work-around:

There is no work-around for this bug.

Example:

The following program will produce a PROGRAMMER ERROR:

```
proc crosstab data=mydata
              filetype=sudxport
              design=brr
              repdata=myrepdata;
    repwgt wtpqrp1--wtpqrp52 / adjfay=2.0408;
    weight wtpfqx6;
    subgroup hssex hab1mi poverty hat28mi;
    levels 2 5 2 3;
    tables hat28mi * hssex *  poverty;
    idvar id_new;
    risk/all;
```

System	Release Reported	Release Fixed
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

340. ALL Procedures: In Standalone SUDAAN formats defined in a LEVFILE are not assigned to the variables if the FORMAT statement is used.

Description:

To supply labels for the levels of one or more variables on SASXPORT files, users should use the LEVFILE option on the PROC statement to identify the file containing the format definitions and then use the FORMAT/RFORMAT statement to assign formats to specific variables. This works when the RFORMAT statement is used, but SUDAAN fails to assign the formats when the FORMAT statement is used.

This does not affect SAS-Callable SUDAAN, only Standalone SUDAAN.

Work-around:

Use the RFORMAT statement instead of the FORMAT statement to assign formats to the variables.

Example:

This example attempts to assign the YESNO format to the DARE variable. The LEVFILE option tells SUDAAN where to find the YESNO format. The FORMAT statement assigns the YESNO format to the DARE variable:

```
proc crosstab data="mydata.stx"
              filetype=sasxport
              levfile="formats.xpt"
              design=srs;
    class dare;
    table dare;
    format dare yn.;
    print nsum / style=nchs;
```

However, SUDAAN does not display labels for the values of the DARE variable:

```
-------------------------------
DARE Program           Sample
                       Size
-------------------------------
Total                     1525
1.000                      822
2.000                      703
-------------------------------
```

After changing FORMAT to RFORMAT, SUDAAN displays the the labels "Yes" and "No" for the levels of the DARE variable:

```
-------------------------------
DARE Program           Sample
                       Size
-------------------------------
Total                     1525
Yes                        822
No                         703
```

System	Release Reported	Release Fixed
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

341. All Procedures: Using the NEWVAR statement to create a variable that already exists on the input file in combination with the NOTSORTED option on the PROC statement causes a PROGRAMMER ERROR.

Description:

When the NOTSORTED option is specified on the PROC statement and a NEWVAR statement is used to create a new variable with the same name as a variable on the data file, SUDAAN will halt with a PROGRAMMER ERROR instead of completing the requested analysis.

Work-around:

There are two work-arounds. First, sort the input file prior to running the procedure rather than using the NOTSORTED option. Second, specify a unique variable name on the NEWVAR statement.

Example:

This program will terminate with a PROGRAMMER ERROR because the momrace variable is on the data file and the NOTSORTED option has been specified:

```
proc regress data=wicwage filetype=sudxport notsorted;
    nest stratum;
    weight analwgt1;
    class educ;
    newvar momrace : if racemom in (1, 2, 3) then momrace = 1
                     else momrace = 2;
    model babywgt = educ age;
    output / predicted=all filename=temp filetype=sudxport replace;
```

This program will run successfully because the data file is sorted prior to running REGRESS and the NOTSORTED option is not used:

```
proc records data=wicwage filetype=sudxport noprint;
    sortby stratum;
    output / filename=wicwage_sorted filetype=sudxport replace;

proc regress data=wicwage_sorted filetype=sudxport;
    nest stratum;
    weight analwgt1;
    class educ;
    newvar momrace : if racemom in (1, 2, 3) then momrace = 1
                     else momrace = 2;
    model babywgt = educ age;
    output / predicted=all filename=temp filetype=sudxport replace;
```

This program will run successfully because the NEWVAR statement creates a unique new variable:

```
proc regress data=wicwage filetype=sudxport notsorted;
    nest stratum;
    weight analwgt1;
    class educ;
    newvar momrace2 : if racemom in (1, 2, 3) then momrace2 = 1
```

```
                 else momrace2 = 2;
model babywgt = educ age;
output / predicted=all filename=temp filetype=sudxport replace;
```

System	Release Reported	Release Fixed
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

342. LOGLINK and REGRESS Procedures: DATA ERROR when the weighted sum of the dependent variable is negative or zero

Description:

SUDAAN generates a DATA ERROR when the weighted sum of the dependent variable is negative or zero in the REGRESS procedure, or when the weighted sum of the dependent variable is zero in the LOGLINK procedure. (LOGLINK doesn't support negative values of the dependent variable.) The DATA ERROR message is "There are no valid observations for this analysis". Analyses such as these are valid, however, so SUDAAN should produce results instead of printing a DATA ERROR.

Work-around:

There is no work-around for this bug. It will be fixed in Release 11.0.1 of SUDAAN, but you may contact SUDAAN Technical Support to request a patched version of SUDAAN if you need it fixed sooner.

Example:

In this example, the DIFF variable is negative on most records, and the weighted sum of DIFF is negative. If you want to run a regression analysis with DIFF as the dependent variable, your SUDAAN code may look like this:

```
proc regress data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class race gender;
    model diff = race gender;
    run;
```

However, instead of running the analysis, SUDAAN prints a DATA ERROR:

```
DATA ERROR
 in Request 79
:
There are no valid observations for this analysis.

SUDAAN processing halted.
```

System	Release Reported	Release Fixed
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

343. All Procedures: An error generated by one value of the variables listed in the BY statement prevents SUDAAN from producing results for the remaining values of the BY variables.

Description:

The BY statement (RBY in SAS-callable SUDAAN) instructs SUDAAN to perform a separate analysis for each subpopulation specified by the values of the variables listed in the BY statement. If SUDAAN encounters an error generated by one value in one of the variables listed in the BY statement, it halts and does not attempt to perform the analysis for the remaining values of the BY variable.

Work-around:

Correcting the error in the affected value of the BY variable should allow SUDAAN to process the remaining values from the BY variable. If correcting the error isn't feasible, then there is no work-around other than removing the BY statement or setting the conflicting value to missing. To emulate BY processing in the absence of the BY statement, perform multiple runs of the procedure with a SUBPOPN or SUBPOPOX statement for each value of the BY variables.

Example:

Suppose you want to fit a logistic regression model using data from each of four regions (region=1,..,4). You might use the RBY statement in the following way:

```
proc rlogist data=mydata design=wr;
    rby region;
    nest stratum psu;
    weight wt;
    class age exercise smoke;
    model obese = age exercise smoke;
    print nsum;
run;
```

If for some reason SUDAAN detects an error when fitting the model for region 1 (maybe because the OBESE variable is missing for every record in region 1), SUDAAN will halt before attempting to fit the model for the other three regions. To fit the model to the other regions, you could write a separate procedure for each region or define a new region variable, region_new using the NEWVAR statement (new to SUDAAN 11.0.0) or another data manipulation progam.

For example, the following would fit the model for region 2:

```
proc rlogist data=mydata design=wr;
    subpopx region = 2;
    nest stratum psu;
    weight wt;
    class age exercise smoke;
    model obese = age exercise smoke;
    print nsum;
run;
```

The same result can be obtained with the following code which fits a model for regions 2 to 4 using the recoded variable region_new:

```
proc rlogist data=mydata design=wr;
    newvar region_new: if region in (2,3,4) then region_new=region
                       else region_new=.;
    rby region_new;
    nest stratum psu;
    weight wt;
    class age exercise smoke;
    model obese = age exercise smoke;
    print nsum;
run;
```

System	**Release Reported**	**Release Fixed**
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

344. KAPMEIER, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, SURVIVAL, WTADJUST, and WTADJX Procedures: System error when computing some variance-derived statistics when only 1 PSU in a sampling stratum

Description:

When the input dataset includes a stratum containing just one PSU (a scenario requiring the MISSUNIT option on the NEST statement), SUDAAN modeling procedures terminate with a system error if certain keywords are requested on the PRINT or OUTPUT statement. The affected keywords are those derived from the variance of a total statistic. This includes some, but not all, of the variance, standard error, confidence interval, and test statistic keywords.

The error message is produced by the operating system, not by SUDAAN. As a result, the message is different on different platforms. It may be described as an access violation, an exception, or a segmentation fault. Another possibility is that a dialog may appear saying SAS or SUDAAN has encountered a problem and needs to close.

The SUDAAN modeling procedures affected are KAPMEIER, LOGISTIC (RLOGIST), LOGLINK, MULTILOG, SURVIVAL, WTADJUST, and WTADJX.

Work-around:

There is no work-around for this bug.

Example:

The following examples produce a system error if the MYDATA dataset includes a stratum with only one PSU. SEPRDMRG and LOWHR/UPHR are keywords that trigger the calculation of the variance of a total statistic.

```
proc rlogist data=mydata design=wr;
    nest stratum psu / missunit;
    weight wt;
    model asthma = year;
    predmarg year / year=(0 1 2 3 4);
    print predmrg seprdmrg;
run;

proc survival data=mydata design=wr;
    nest stratum psu / missunit;
    weight weight;
    event cancer;
    class sex race;
    model age ageend = sex race / intervals=100;
    print hr lowhr uphr;
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

345. All Procedures: Syntax error generated when using minus signs on NEWVAR, SUBPOPX, PARAMETER, and X statements

Description:

Under certain conditions, SUDAAN produces a syntax error when a minus sign is used in an expression included in specific statements. This affects the NEWVAR and SUBPOPX statements available in all procedures and the PARAMETER and X statements in the VARGEN procedure. The syntax error message is similar to the following: Tokens '-' and 'Y' cannot be adjacent.

Work-around:

Multiply by -1 instead of using the minus sign. This workaround applies when the minus sign precedes a variable to denote a "change of the sign of the variable" or used as a subtraction operator. For example, replace -Y with -1*Y, and replace X – Y with X + -1*Y.

Example:

The following two examples produce a syntax error when SUDAAN encounters the minus sign:

```
proc descript data=mydata design=wr;
    nest stratum psu;
    weight wt;
    subpopx diff > x - xold;
    var score;
    print mean semean;
run;

proc vargen data=mydata method=forward design=wr step=.001;
    nest stratum psu;
    weight wt;
    xmean expage: exp(-agegrp);
    tables _one_;
    print estim seestim;
run;
```

Applying the work-around prevents the syntax error:

```
proc descript data=mydata design=wr;
    nest stratum psu;
    weight wt;
    subpopx diff > x + -1*xold;
    var score;
    print mean semean;
run;

proc vargen data=mydata method=forward design=wr step=.001;
    nest stratum psu;
    weight wt;
    xmean expage: exp(-1*agegrp);
    tables _one_;
    print estim seestim;
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

346. CROSSTAB, DESCRIPT, LOGISTIC (RLOGIST), RATIO, VARGEN, WTADJUST, and WTADJX Procedures: SYNTAX ERROR when variables are specified using dash notation on TABLE statement

Description:

When you want to specify a series of variable names of the form VAR1 VAR2 ... VAR*n* on the TABLE statement, you can use the dash notation as a shortcut by specifying the variables as VAR1-VAR*n*. However, SUDAAN produces a SYNTAX ERROR when the dash notation is used.

Work-around:

Specify all the variables individually on the TABLE statement instead of using the dash notation.

Example:

The following procedure produces a SYNTAX ERROR because of the dash notation on the TABLE statement:

```
proc crosstab data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class income1-income5;
    table income1-income5;
    print totper setot;
run;
```

Specify each variable individually on the TABLE statement to avoid the error:

```
    class income1-income5;
    table income1 income2 income3 income4 income5;
```

Note that the dash notation on the CLASS statement doesn't cause an error and therefore doesn't need to be modified.

System	**Release Reported**	**Release Fixed**
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

347. WTADJUST and WTADJX Procedures: Incorrect values for the CNTLTOTAL, DIFFWT, and MARGADJ keywords when the REFLEVEL statement is used

Description:

When the REFLEVEL statement is used in the WTADJUST procedure, SUDAAN computes incorrect values for the CNTLTOTAL keyword. It swaps the control totals for the default reference level and the requested reference level. The DIFFWT keyword, defined as TOTFINAL – CNTLTOTAL, and the MARGADJ keyword, defined as CNTLTOTAL/TOTTRIM, are also computed incorrectly. The same problem exists in the WTADJX procedure, although the affected keywords have slightly different names: CNTLTOTALZ, DIFFWTZ, and MARGADJZ and similar definitions.

Work-around:

Instead of using the REFLEVEL statement, recode the variables you would list in the REFLEVEL statement so that the desired reference levels correspond to the highest numeric level, which is the default for SUDAAN. You can do this in a separate data step before running WTADJUST or WTADJX procedure, or, starting in 11.0.0, you can do it within the procedure using the NEWVAR statement. An example using NEWVAR is provided in the **Example** section below.

Example:

In this example, the PROGRAM variable is used as a categorical model covariate. It has 3 levels, so by default SUDAAN treats the 3rd level (PROGRAM=3) as the reference level:

```
proc wtadjust data=mydata design=wr adjust=nonresponse;
    nest stratum psu;
    weight wt;
    subgroup PROGRAM;
    levels 3;
    model pass = PROGRAM;
    print cntltotal tottrim margadj;
    run;
```

The printed results might look something like this:

```
----------------------------------------------------------------

Independent
  Variables and                     Sum of
  Effects                           Trimmed        Marginal
                       Control      Weights Over   Weight
                       Totals       Respondents    Adjustment
----------------------------------------------------------------
Intercept                  469.00         184.00         2.5489
PROGRAM
  1                        271.00         102.00         2.6569
  2                         95.00          44.00         2.1591
  3                        103.00          38.00         2.7105
----------------------------------------------------------------
```

Now suppose you want the reference level to be PROGRAM=2. Adding a REFLEVEL statement overrides SUDAAN's default choice of reference level:

```
proc wtadjust data=mydata design=wr adjust=nonresponse;
    nest stratum psu;
    weight wt;
    subgroup PROGRAM;
    levels 3;
    model pass = PROGRAM;
    reflevel PROGRAM=2;
    print cntltotal tottrim margadj;
    run;
```

Simply changing the reference level should not affect any of the three keywords being printed in this example. However the output shows that the control totals for levels 2 and 3 have been switched, the values under the "Sum of the trimmed weights Over Respondents" column remain the same, and consequently the values under the "Marginal weight adjustments" column are also different:

```
----------------------------------------------------------------

Independent
  Variables and                     Sum of
  Effects                           Trimmed        Marginal
                       Control      Weights Over   Weight
                       Totals       Respondents    Adjustment
----------------------------------------------------------------
Intercept                    469.00         184.00       2.5489
PROGRAM
  1                          271.00         102.00       2.6569
  2                          103.00          44.00       2.3409
  3                           95.00          38.00       2.5000
----------------------------------------------------------------
```

Using CROSSTAB to compute the control totals independently:

```
proc crosstab data=mydata design=wr;
    nest stratum psu;
    weight wt;
    subgroup program;
    levels 3;
    table program;
    print wsum / style=nchs;
    run;
```

shows that the control totals computed by WTADJUST *without* the REFLEVEL statement are the correct ones:

```
---------------------------------
PROGRAM                Weighted
                       Size
---------------------------------
Total                      469.00
1                          271.00
2                           95.00
3                          103.00
---------------------------------
```

One work-around is to create a new PROGRAM variable where levels 2 and 3 have been switched. Then SUDAAN will treat the new level 3 (the original level 2) as the reference level by default, so the REFLEVEL statement is no longer needed:

```
proc wtadjust data=mydata design=wr adjust=nonresponse;
```

```
newvar program_new:
         if program = 1 then program_new = 1
    else if program = 2 then program_new = 3
    else if program = 3 then program_new = 2
    else                      program_new = .;
nest stratum psu;
weight wt;
subgroup program_new;
levels 3;
model pass = program_new;
print cntltotal tottrim margadj;
run;
```

The printed results show that the control totals and marginal adjustments match the values obtained by WTADJUST originally, when neither the NEWVAR nor the REFLEVEL statements were used (keeping in mind that PROGRAM_NEW=2 is equivalent to PROGRAM=3 and vice versa):

```
----------------------------------------------------------------

Independent
  Variables and                     Sum of
  Effects                           Trimmed        Marginal
                       Control      Weights Over   Weight
                       Totals       Respondents    Adjustment
----------------------------------------------------------------
Intercept                    469.00         184.00        2.5489
PROGRAM_NEW
  1                          271.00         102.00        2.6569
  2                          103.00          38.00        2.7105
  3                           95.00          44.00        2.1591
----------------------------------------------------------------
```

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

348. All Procedures: Access violation when REPLACE or APPEND not specified for RTF print format.

Description:

If users choose to print in RTF format but don’t specify the REPLACE or APPEND options on the PRINT statement, SUDAAN will produce an access violation error and halt.

Work-around:

The work-around is to provide the REPLACE or APPEND options on the PRINT statement.

Example:

This example produces an access violation error:

```
proc vargen data=mydata design=wr;
    nest stratum psu;
    weight wt;
    xmean x1: ((myvar1));
    print /filetype=RTF filename=”myrtf”;
run;
```

The following two examples do not produce programmer errors:

```
proc vargen data=mydata design=wr;
    nest stratum psu;
    weight wt;
    xmean x1: myvar1;
    print /filetype=RTF filename=”myrtf” REPLACE;
run;

proc vargen data=mydata design=wr;
    nest stratum psu;
    weight wt;
    xmean x1: 1*((myvar1));
    print /filetype=RTF filename=”myrtf” APPEND;
run;
```

System	**Release Reported**	**Release Fixed**
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

349. LOGLINK, MULTILOG and REGRESS Procedures: Unable to output keywords from the VARIANCE group

Description:

When the VARIANCE group is specified on the OUTPUT statement in the REGRESS, LOGLINK, or MULTILOG procedures, SUDAAN issues the SEMANTIC ERROR "DDF, ESTIMATE and COVAR are required for COVAR output". It issues the same error if certain keywords in the VARIANCE group are explicitly listed on the OUTPUT statement. For example, if the DDF keyword (a member of the VARIANCE group) is explicitly listed on the OUTPUT statement, SUDAAN issues a different SEMANTIC ERROR: "Keyword DDF is not available on PRINT or OUTPUT with the current options".

Work-around:

There is no work-around for this bug. It will be fixed in Release 11.0.1 of SUDAAN, but you may contact SUDAAN Technical Support to request a patched version of SUDAAN if you need it fixed sooner.

Example:

In this example, the VARIANCE group is requested on the OUTPUT statement:

```
proc regress data=mydata design=wr noprint;
    nest stratum psu;
    weight wt;
    class veg;
    model bmi = veg;
    output / variance=default filename=variance_out replace;
    run;
```

However, instead of running the analysis, SUDAAN prints a SEMANTIC ERROR:

```
SEMANTIC ERROR
:
DDF, ESTIMATE and COVAR are required for COVAR output

SUDAAN processing halted.
```

If specific keywords are requested instead of the entire VARIANCE group, and one of those is the DDF keyword:

```
proc regress data=mydata design=wr noprint;
    nest stratum psu;
    weight wt;
    class veg;
    model bmi = veg;
    output covar ddf / filename=covar_out replace;
    run;
```

then SUDAAN prints this SEMANTIC ERROR:

```
SEMANTIC ERROR
:
Keyword DDF is not available on PRINT or OUTPUT with the current options
Please refer to your SUDAAN manual for a discussion of conditions which
may affect the
```

```
        availability of DDF on the OUTPUT statment

        SUDAAN processing halted.
```

System	**Release Reported**	**Release Fixed**
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

350. LOGISTIC (RLOGIST), WTADJUST and WTADJX Procedures: Unable to output the default keywords from the MEANCOV, PCTCOV, RHATCOV, and TOTCOV groups

Description:

When the MEANCOV=DEFAULT group is requested on the OUTPUT statement in the RLOGIST procedure, SUDAAN issues the SEMANTIC ERROR "DDF, ESTIMATE and COVAR are required for COVAR output". The same error is issued when PCTCOV=DEFAULT, RHATCOV=DEFAULT, or TOTCOV=DEFAULT is requested. The same error is also issued in the WTADJUST and WTADJX procedures.

Work-around:

SUDAAN will run without producing the SEMANTIC ERROR if you request MEANCOV=ALL instead of MEANCOV=DEFAULT (and similarly for the other output groups). Another work-around is to explicitly add COVMEAN (or the analogous covariance keyword in the other output groups) to the OUTPUT statement.

Example:

In this example, MEANCOV=DEFAULT is requested on the OUTPUT statement:

```
proc rlogist data=mydata design=wr noprint;
    nest stratum psu;
    weight wt;
    class veg;
    model bmi = veg;
    var smoke;
    output / meancov=default filename=meancov_out replace;
    run;
```

Instead of running the analysis, SUDAAN prints a SEMANTIC ERROR:

```
SEMANTIC ERROR: DDF, ESTIMATE and COVAR are required for COVAR output

SUDAAN processing halted.
```

SUDAAN will run the analysis without producing the error if the OUTPUT statement is changed to request all the MEANCOV keywords:

```
output / meancov=all filename=meancov_out replace;
```

or if COVMEAN is requested explicitly in addition to the default keywords:

```
output covmean / meancov=default filename=meancov_out replace;
```

System	Release Reported	Release Fixed
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

351. LOGISTIC (RLOGIST), LOGLINK, MULTILOG, REGRESS, WTADJUST and WTADJX Procedures: Fatal memory allocation error

Description:

SAS-callable SUDAAN is limited to the amount of physical memory specified by SAS's MEMSIZE option. However, SUDAAN ignores SAS's MEMSIZE option and instead assumes it can use all the physical memory available to the operating system. Typically, the amount available to SAS is less than the amount available to the operating system. Therefore, SUDAAN overestimates the amount of available memory. As a result, it attempts to allocate more memory than is available, sometimes leading to a fatal error with a message similar to: "Not enough memory for this job. Unable to allocate a block of size 580.82 MB. 41.78 MB already used."

This bug does not affect standalone SUDAAN.

Work-arounds:

There are three potential work arounds for this bug:

1. On the PROC statement, set SUDAAN's USEVMEM option to 1. This forces SUDAAN to minimize the amount of physical memory it consumes, therefore increasing the probability that it needs less memory than the amount specified by SAS's MEMSIZE option. Setting USEVMEM=1 may have the side effect of making your program run more slowly than it would using the other work arounds.
2. Set SUDAAN's USEVMEM option to a value that is lower than the value of SAS's MEMSIZE option. This prevents SUDAAN from attempting to use more physical memory than the amount available to SAS. To compute a reasonable USEVMEM value, find the MEMSIZE value by submitting the code `proc options option=memsize; run;`, convert to megabytes by dividing the result by 1024*1024, and provide yourself a small buffer for SAS's overhead by multiplying the result by a fraction such as 0.9.
3. Change the value of SAS's MEMSIZE option to MAX. This option should be changed by editing the sasv9.cfg file or by adding it to the command line used to invoke SAS. The effect of this change is to make all of the system's memory available to SAS. Therefore, this may not be a suitable work around if you plan to run large jobs in SAS while simultaneously doing other work on your computer.

Example:

Suppose you're trying to run a logistic regression using a large number of covariates. For example:

```
proc rlogist data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class x1-x1000 / nofreq;
    model y = x1-x1000;
    print beta sebeta;
run;
```

Such a job may consume a large amount of memory, maybe more than the amount specified by SAS's MEMSIZE option. If so, SUDAAN will issue an error message like the following:

```
Not enough memory for this job
Unable to allocate a block of size 933.46 MB
63.40 MB already used.
FATAL ERROR in SUDAAN
```

The quickest work around is to try adding the USEVMEM=1 option:

```
proc rlogist data=mydata design=wr usevmem=1;
```

An alternative is to set USEVMEM to a value less than the MEMSIZE value. First find the MEMSIZE value using:

```
proc options option=memsize; run;
```

Suppose it returns 1073741824. Using the formula mentioned in the Word-Arounds section, this translates to a USEVMEM value of 1073741824 / (1024*1024) * 0.9 = 921.

```
proc rlogist data=mydata design=wr usevmem=921;
```

The final work around is to find the MEMSIZE option in your sasv9.cfg and change it to MAX:

```
-MEMSIZE MAX
```

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

352. ALL Procedures: PROGRAMMER ERROR when a variable on the RFORMAT statement does not appear on the dataset

Description:

When a variable specified on the RFORMAT statement does not appear on the dataset (for example, when the variable name is misspelled on the RFORMAT statement), SUDAAN correctly issues a FILE ERROR saying it couldn't find that variable. However, instead of terminating, SUDAAN continues and eventually issues a PROGRAMMER ERROR.

Work-around:

Review all the variables listed on RFORMAT statements. If any of them don't appear on the dataset, either remove them or fix their names.

Example:

In this example, the variable DARE is on the input data set, but it is misspelled as DSRE on the RFORMAT statement:

```
RFORMAT INTCIG12 DSRE YN.;
```

SUDAAN correctly prints a FILE ERROR to the log file:

```
8   RFORMAT  INTCIG12 DSRE YN.;

FILE ERROR
 in Statement 8
:
(Message 105)
Variable DSRE not found
```

However, it then additionally prints a PROGRAMMER ERROR:

```
PROGRAMMER ERROR
 in Object 0
:
NOTICE:
You may have found a bug in the SUDAAN software.
Please contact Research Triangle Institute with
complete information about this job so that we can
attempt to locate and correct the problem.


(Message 503)
Current object is not expected type
Offset is 0 object type is NULL (0)
Expected type is 12
FATAL ERROR in SUDAAN
```

Correcting the misspelling on the RFORMAT statement fixes the problem:

```
RFORMAT INTCIG12 DARE YN.;
```

System	**Release Reported**	**Release Fixed**
Windows	11.0.0	11.0.1
Solaris	n/a	n/a
Linux	n/a	n/a

353. VARGEN Procedure: A PROGRAMMER ERROR is encountered when using PROC statement option DESIGN=SRS.

Description:

SUDAAN produces a PROGRAMMER ERROR for the VARGEN procedure if sampling design option SRS is specified on the PROC statement.   SUDAAN does not produce a PROGRAMMER ERROR for any other design options.

Work-around:

There is no work-around for this bug. It will be fixed in Release 11.0.1 of SUDAAN, but you may contact SUDAAN Technical Support to request a patched version of SUDAAN if you need it fixed sooner. The closest result to SRS will be obtained by specifying DESIGN=WR with the _ONE_ keyword on both the NEST and WEIGHT statements. Results will be asymptotically equivalent to DESIGN=SRS.

Example:

In this example, design option SRS is specified:

```
proc vargen data=mydata method=forward design=srs step=.001;
    xmean var1: exp(-1*var2);
    tables _one_;
    print estim seestim;
    run;
```

SUDAAN will produce a programmer error and halt.

If the design option is something other than SRS:

```
proc vargen data=mydata method=forward design=wr step=.001;
    nest _one_;
    weight _one_;
    xmean var1: exp(-1*var2);
    tables _one_;
    print estim seestim;
    run;
```

SUDAAN will proceed without error. Note that DESIGN=WR with the _ONE_ keyword on both NEST and WEIGHT statements produce results asymptotically equivalent to DESIGN=SRS.

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

354. LOGLINK Procedure: Floating point error when the dependent variable has the same value on every record.

Description:

When the value of the dependent variable is the same on every record, the LOGLINK procedure sometimes halts without performing the analysis. SAS-callable SUDAAN halts with the error messages "ERROR: Floating Point Zero Divide" and "ERROR: Termination due to Floating Point Exception". Standalone SUDAAN halts with the error message "Argument singularity error in SUDAAN function log".

On Windows platforms, when the value of the dependent variable is 0 on every record, this bug is masked by a different bug (bug #342: DATA ERROR when the weighted sum of the dependent variable is negative or zero).

Work-around:

First verify that you have set up your analysis the way you intended. If the dependent variable has the same value on every record, this is a warning sign that there may be a mistake somewhere. For example, the data in your input dataset may be incorrect, you may have specified the wrong dependent variable, or your SUBPOPN/SUBPOPX statement may have restricted the analysis to a smaller subset of the records than you expected.

To work around the bug, add an INITPARM statement with a non-zero value for the intercept's initial beta.

Examples:

In this example, suppose the value of the COUNT variable is 1 on every record. The following procedure will produce a floating point error:

```
proc loglink data=mydata design=srs;
    class gender;
    model count = gender;
    print beta;
run;
```

The work-around is to initialize the intercept's beta to a non-zero value using the INITPARM statement:

```
proc loglink data=mydata design=srs;
    class gender;
    model count = gender;
    initparm 1 0;
    print beta;
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

355. LOGLINK, MULTILOG, and REGRESS Procedures: Unable to print the DDF keyword

Description:

SUDAAN does not print the DDF keyword when it is listed on the PRINT statement explicitly or when it is included implicitly using VARIANCE=ALL or VARIANCE=DEFAULT. When DDF is listed explicitly, SUDAAN issues the SEMANTIC ERROR: "Keyword DDF is not available on PRINT or OUTPUT with the current options". When DDF is included as part of the VARIANCE group, SUDAAN prints the other keywords in the group but omits DDF.

Work-around:

There is no work-around for this bug. It will be fixed in Release 11.0.1 of SUDAAN, but you may contact SUDAAN Technical Support to request a patched version of SUDAAN if you need it fixed sooner.

Example:

In this example, the VARIANCE group is requested on the PRINT statement:

```
proc regress data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class veg;
    model bmi = veg;
    print / variance=default;
    run;
```

The default set of keywords in the VARIANCE group consists of DDF, CHECK, RANK, BETA, COVAR, IDMPTNT, and MODELCOV. SUDAAN runs the analysis and prints all of these keywords except for DDF.

If the procedure is changed to request DDF specifically:

```
proc regress data=mydata design=wr;
    nest stratum psu;
    weight wt;
    class veg;
    model bmi = veg;
    print ddf;
    run;
```

then SUDAAN prints this SEMANTIC ERROR:

```
SEMANTIC ERROR
:
Keyword DDF is not available on PRINT or OUTPUT with the current options
Please refer to your SUDAAN manual for a discussion of conditions which
may affect theavailability of DDF on the OUTPUT statment

SUDAAN processing halted.
```

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

357. All Procedures: FILE ERROR when the definition of a NEWVAR variable includes a previous NEWVAR variable.

Description:

When defining a new variable using the NEWVAR statement, the user should be able to write an expression that is a function of any existing variable. Existing variables include all the variables on the input dataset as well as variables created by previous NEWVAR statements. However, when a NEWVAR statement includes a variable created by a previous NEWVAR statement, SUDAAN halts with a FILE ERROR stating that the variable cannot be found.

Work-around:

One work-around is to create the new variables outside of SUDAAN prior to running the SUDAAN analysis.

Another work-around is to rewrite the NEWVAR expressions to avoid the use of intermediate NEWVAR variables, instead using only those variables that appear on the input dataset. Although the NEWVAR expressions may become more complicated, it should be possible to replace any occurrence of an intermediate variable with the expression used when defining that intermediate variable.

Example:

In the following example, suppose the variables X and Y appear in your input dataset, and you wish to find the mean of X+Y and $(X+Y)^2$. Because these values do not appear as distinct variables on the input dataset, you decide to create them with the NEWVAR statement:

```
proc descript data=mydata design=wr;
    nest stratum psu;
    weight wt;
    newvar z:   z = x + y;
    newvar z2: z2 = z*z;
    var x y z z2;
    print mean semean;
run;
```

However, because Z2 is defined in terms of Z, and Z was created by a NEWVAR statement, the bug described here prevents this code from running successfully. A work-around is to rewrite Z2 such that its definition involves only the variables X and Y, not Z:

```
    newvar z2: z2 = (x+y)*(x+y);
```

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

358. VARGEN Procedure: Using an expression containing nested parentheses may cause a PROGRAMMER ERROR.

Description:

The VARGEN procedure allows users to define functions for all X-Statements, XU-Statements and the PARAMETER statement. If the function starts with nested parentheses, VARGEN will produce a PROGRAMMER ERROR.

Work-around:

One work-around is to eliminate unnecessary nested parentheses.

Another work-around is to rearrange the expression so that it does not start with nested parentheses. For example, add "1 *" or "0 +" in front of the expression.

Example:

This example produces a PROGRAMMER ERROR:

```
proc vargen data=mydata design=wr;
    nest stratum psu;
    weight wt;
    xmean x1: ((myvar1));
    print ;
run;
```

The following two examples do not produce a PROGRAMMER ERROR:

```
proc vargen data=mydata design=wr;
    nest stratum psu;
    weight wt;
    xmean x1: myvar1;
    print ;
run;

proc vargen data=mydata design=wr;
    nest stratum psu;
    weight wt;
    xmean x1: 1*((myvar1));
    print ;
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

359. All Procedures: Floating point error when class variable contains large numbers for SAS-Callable SUDAAN.

Description:

For SAS-Callable SUDAAN, if a CLASS statement variable contains large values that exceed the limit of system integer representation, a floating point error will be generated. Standalone SUDAAN will not generate a floating point error, but the computational result will be incorrect.

Work-around:

One work-around is to scale down the values of class variable that are larger than integer `2147483647`.

Example:

This example produces a floating point error if the CLASS variable MY_ID contains values that are larger than 2,147,483,647.

```
proc regress data=mydata design=wr;
    nest stratum psu;
    weight WEIGHT;
    class  my_id;
    model myresult = my_id;
    print beta sebeta;
    title “SUDAAN Run”;
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

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

361. LOGISTIC (RLOGIST) and LOGLINK Procedures: Additional iteration in convergence algorithm for exchangeable working correlations

Description:

When assuming exchangeable working correlations in the LOGISTIC (RLOGIST) and LOGLINK procedures (that is, when either the R=EXCHANGEABLE or RSTEPS=*n*>0 parameter is specified), SUDAAN sometimes continues the convergence algorithm in the exchangeable steps for one iteration beyond the number required for convergence. It computes the additional iteration whenever the MAXXITER parameter is set to a value greater than *k*, where *k* is the number of iterations required for convergence in the exchangeable step.

The impact of the additional iteration is a small change in the BETA and SEBETA keywords as well as any keywords derived from them. Because the model is convergent, the additional iteration should produce a slightly more accurate approximation of these keywords.

Work-around:

If *k* is the number of iterations required for convergence of the exchangeable steps, then set MAXXITER=*k* to force SUDAAN to stop iterating after exactly *k* iterations. To find the value of *k*, first run the procedure with a larger value of MAXXITER, and then subtract 1 from the reported number of iterations.

Example:

Consider the following SUDAAN analysis:

```
proc rlogist data=mydata design=wr r=exchangeable maxxiter=10;
    nest stratum psu;
    weight wt;
    class age gender;
    model resp = age gender;
    print beta sebeta;
run;
```

Suppose the exchangeable step converges in 5 iterations. If you run the RLOGIST procedure with MAXXITER=10, SUDAAN reports that the exchangeable step converges in 6 iterations:

```
Step 1 parameters have converged in 6 iterations.
```

If you re-run the procedure with MAXXITER=5, SUDAAN now reports that the exchangeable step converges in 5 iterations:

```
Step 1 parameters have converged in 5 iterations.
```

The BETAs and SEBETAs produced by these analyses will be different, but the difference should be very small. The following table demonstrates the magnitude of the difference for one particular example:

Iterations	BETA	SEBETA
5	2.8490728214628680	0.0955830138706311

Iterations	BETA	SEBETA
6	2.8490728210778776	0.0955830137995846

System	Release Reported	Release Fixed
Windows	11.0.0	not fixed
Solaris	11.0.0	not fixed
Linux	11.0.0	not fixed

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

363. VARGEN Procedure: PROGRAMMER ERROR when contrasts and multiple tables are requested for BRR designs

Description:

The VARGEN procedure produces a PROGRAMMER ERROR when all of the following conditions are met:

- The BRR design is used.
- One or more contrasts are defined. Contrasts may be defined using the CONTRAST, DIFFVAR, PAIRWISE, or POLY statements.
- More than one term is listed on the TABLE statement.

Work-around:

Instead of listing multiple terms on the TABLE statement in a single procedure call, rewrite the analysis as a series of procedure calls, each having a single term on the TABLE statement.

Example:

The following SUDAAN procedure meets all three conditions for this bug and will therefore produce a PROGRAMMER ERROR:

```
proc vargen data=mydata design=brr;
    weight wt;
    repwgt repwt1-repwt24;
    class gender educ agegrp race;
    table gender educ*agegrp;
    pairwise race;
    xmean mean_bmi: bmi;
    print estim seestim;
    run;
```

To avoid the PROGRAMMER ERROR, split the VARGEN analysis into two separate analyses. Include just one term on the TABLE statement for each analysis:

```
proc vargen data=mydata design=brr;
    weight wt;
    repwgt repwt1-repwt24;
    class gender educ agegrp race;
    table gender;
    pairwise race;
    xmean mean_bmi: bmi;
    print estim seestim;
    run;

proc vargen data=mydata design=brr;
    weight wt;
    repwgt repwt1-repwt24;
    class gender educ agegrp race;
    table educ*agegrp;
    pairwise race;
    xmean mean_bmi: bmi;
    print estim seestim;
    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

364. VARGEN Procedure: Incorrect standard errors for subpopulation estimates with replicate designs

Description:

For the BRR and replicate-weight jackknife designs, the VARGEN procedure produces incorrect standard errors for subpopulation estimates. Subpopulation estimates may be obtained by using the SUBPOPN, SUBPOPX, or BY (RBY) statement. This bug does not affect the delete-1 jackknife design.

Work-around:

To produce subpopulation estimates in VARGEN with the BRR or replicate-weight jackknife design, avoid using the SUBPOPN, SUBPOPX, or BY statement. Instead, first create a new input dataset containing only the observations in the desired subpopulation. For a BY-group analysis, create one new input dataset for each BY-group. Then run VARGEN on the new input dataset(s).

Please note that this bug and work-around apply only to the replicate designs. For replicate designs only, deleting observations from the input dataset is equivalent to establishing a subpopulation using the SUBPOPN or SUBPOPX statement. The same is not true for Taylor series designs such as WR. For Taylor series designs, users are advised not to delete observations from the dataset to produce subpopulation estimates in SUDAAN.

Examples:

The following VARGEN procedure specifies the BRR design and uses the SUBPOPX statement to restrict the analysis to observations from the year 2010. If the dataset includes additional observations from years other than 2010, then the standard errors will be incorrect.

```
proc vargen data=mydata design=brr;
    weight wt;
    repwgt repwt1-repwt24;
    subpopx year eq 2010;
    class gender educ agegrp race;
    table gender educ*agegrp;
    pairwise race;
    xmean mean_bmi: bmi;
    print estim seestim;
    run;
```

VARGEN produces correct standard errors with the BRR design when the input dataset includes observations from the year 2010 only. To achieve this, first subset the input data to the subpopulation before using it with VARGEN:

```
proc records data=mydata noprint;
    subpopx year eq 2010;
    output / filename=mydata_2010 replace;
    run;

proc vargen data=mydata_2010 design=brr;
    weight wt;
    repwgt repwt1-repwt24;
    class gender educ agegrp race;
    table gender educ*agegrp;
    pairwise race;
```

```
          xmean mean_bmi: bmi;
          print estim seestim;
          run;
```

The next example also specifies the BRR design, but this time it uses the RBY statement to produce estimates by gender. VARGEN will compute one set of estimates for the male subpopulation and another set for the female subpopulation. The standard errors for these by-gender estimates will be incorrect.

```
      proc vargen data=mydata design=brr;
          weight wt;
          repwgt repwt1-repwt24;
          rby gender;
          class educ agegrp race;
          table educ*agegrp;
          pairwise race;
          xmean mean_bmi: bmi;
          print estim seestim;
          run;
```

To obtain the correct standard errors, create a separate input dataset for each subpopulation defined by the RBY statement. In this example, one input dataset is needed for males and another for females.

```
      proc records data=mydata noprint;
          subpopx gender eq 1;
          output / filename=mydata_male replace;
          run;

      proc records data=mydata noprint;
          subpopx gender eq 2;
          output / filename=mydata_female replace;
          run;

      proc vargen data=mydata_male design=brr;
          weight wt;
          repwgt repwt1-repwt24;
          class educ agegrp race;
          table educ*agegrp;
          pairwise race;
          xmean mean_bmi: bmi;
          print estim seestim;
          run;

      proc vargen data=mydata_female design=brr;
          weight wt;
          repwgt repwt1-repwt24;
          class educ agegrp race;
          table educ*agegrp;
          pairwise race;
          xmean mean_bmi: bmi;
          print estim seestim;
          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

365. VARGEN Procedure: Incorrect values for VARIABLE on output dataset when contrasts and multiple tables are requested

Description:

Output datasets produced by the VARGEN procedure include a variable named VARIABLE. The value of VARIABLE for a particular record in the output dataset identifies the X-statement or PARAMETER statement that corresponds to the record. For example, if VARIABLE=3, then that record corresponds to the 3rd X-statement or PARAMETER statement defined in the procedure call. SUDAAN assigns incorrect values to VARIABLE for some records when both of the following conditions are met:

- One or more contrasts are defined. Contrasts may be defined using the CONTRAST, DIFFVAR, PAIRWISE, or POLY statements.
- More than one term is listed on the TABLE statement.

Work-around:

Instead of listing multiple terms on the TABLE statement in a single procedure call, rewrite the analysis as a series of procedure calls, each having a single term on the TABLE statement.

Example:

The following VARGEN procedure call defines a single X-statement (XMEAN) and no PARAMETER statements, and so there is one "variable" defined by this procedure call. The value of VARIABLE on the output dataset should be 1 on every record.

```
proc vargen data=mydata design=wr noprint;
    nest stratum psu;
    weight wt;
    subgroup gender smoker veg;
    levels        2      2   2;
    table gender smoker;
    pairwise veg;
    xmean mean_score: score;
    output estim / filename=myout replace;
    run;
```

Because the procedure call requests a contrast with the PAIRWISE statement and two terms are listed on the TABLE statement, it meets the conditions for this bug, causing SUDAAN to assign incorrect values to VARIABLE on the output dataset. In this example, it sets VARIABLE=2 for the last three records. The MYOUT dataset will look something like this (some columns have been omitted for space purposes):

TABLENO	GENDER	SMOKER	VARIABLE	CONTRAST	ESTIM
1	0	-2	1	1	72.7
1	1	-2	1	1	70.3
1	2	-2	1	1	73.8
2	-2	0	**2**	1	72.7
2	-2	1	**2**	1	68.0
2	-2	2	**2**	1	76.1

To work-around the bug, split the VARGEN analysis into two separate analyses. List just one term on the TABLE statement for each analysis:

```
proc vargen data=mydata design=wr noprint;
    nest stratum psu;
    weight wt;
    subgroup gender veg;
    levels        2   2;
    table gender;
    pairwise veg;
    xmean mean_score: score;
    output estim / filename=myout_gender replace;
    run;

proc vargen data=mydata design=wr noprint;
    nest stratum psu;
    weight wt;
    subgroup smoker veg;
    levels        2   2;
    table smoker;
    pairwise veg;
    xmean mean_score: score;
    output estim / filename=myout_smoker replace;
    run;
```

The MYOUT_GENDER dataset will look something like this:

TABLENO	GENDER	VARIABLE	CONTRAST	ESTIM
1	0	1	1	72.7
1	1	1	1	70.3
1	2	1	1	73.8

And the MYOUT_SMOKER dataset will look something like this:

TABLENO	SMOKER	VARIABLE	CONTRAST	ESTIM
2	0	1	1	72.7
2	1	1	1	68.0
2	2	1	1	76.1

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

366. All Procedures: Unexpected error messages when character variables appear on certain statements

Description:

If a character variable appears on a statement that doesn't support character variables, SUDAAN should issue a FILE ERROR and halt. For certain statements, SUDAAN does issue the FILE ERROR as expected, but then it continues and issues one or more additional errors. The additional errors may vary from run to run. The following additional errors have been observed:

- A PROGRAMMER ERROR.
- One or more SEMANTIC ERRORs stating that required statements are missing, even though the statements are clearly present.
- One or more SYNTAX ERRORs stating that a character found in the input is invalid, even though that character doesn't exist in the program.
- A FILE ERROR stating that the [blank] SAS file cannot be opened.

The affected statements are NEST, WEIGHT, TOTCNT, SAMCNT, JOINTPROB, REPWGT, SUBGROUP, and IDVAR.

Work-around:

Do not include character variables on SUDAAN statements. Prior to running the SUDAAN analysis, recode the character variables as numeric variables, and then use the numeric variables in place of the character variables.

Example:

The character variable CHAR_PSU appears on the NEST statement in the following procedure:

```
proc crosstab data=mydata design=wr;
    nest stratum char_psu;
    weight wt;
    class gender smoker;
    table gender*smoker;
    print rowper serow;
    run;
```

After running this procedure, the log file correctly includes a FILE ERROR informing the user that the NEST statement requires numeric variables. It then incorrectly goes on to issue a series of SEMANTIC ERRORs claiming the NEST and WEIGHT statements are missing:

```
FILE ERROR
:
(Message 128)
Variable 'CHAR_PSU' on NEST statement is not numeric.


SEMANTIC ERROR
:
(Message 406)
A WEIGHT statement is required in PROC CROSSTAB


SEMANTIC ERROR
```

```
:
(Message 406)
A NEST statement is required in PROC CROSSTAB

SEMANTIC ERROR
:
The NEST statement is required with the DESIGN=WR parameter on the PROC
statement.

SEMANTIC ERROR
:
The WEIGHT statement is required with the DESIGN=WR parameter on the PROC
statement.
```

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