

DATA ANALYSIS PROTOCOLS FOR THE STATUS NETWORK, Version 1.2

GUIDELINES FOR FLORIDA'S PROBABILISTIC MONITORING NETWORK

Florida Department of Environmental Protection
Division of Environmental Assessment and Restoration
Bureau of Assessment and Restoration Support

**Originally written by Neal A. Doran
Tallahassee, Florida
2007**

**Revised 2008 by Jay Silvanima, Kunle Olumide, & Chris Sedlacek
Revised 2011 by Jay Silvanima, Rick Copeland, & Chris Sedlacek**

TABLE OF CONTENTS

TABLE OF CONTENTS	2
TABLE OF FIGURES	2
INTRODUCTION	3
PART I—THE STATUS NETWORK.....	4
QUESTIONS ADDRESSED BY THE STATUS NETWORK.....	5
PART II—DATA QUALITY ASSESSMENTS.....	5
ERRORS IN THE DATA	5
<i>Outliers</i>	5
<i>Missing Values</i>	6
<i>Detection Limits</i>	6
<i>Qualifier Codes</i>	6
PART III—ANALYSIS PROCEDURES	7
RUNNING THE PROGRAM IN S-PLUS	7
EXPLANATIONS FOR EACH PORTION OF SCRIPT:	10
<i>Plotting CDFs</i>	14
REFERENCES	16
APPENDIX A—EXAMPLE SMALL STREAMS CODE	17
APPENDIX B—DATA QUALIFIERS.....	27
APPENDIX C—EXAMPLE SITEINFO FILE.....	29
APPENDIX D—ADJUST WEIGHT CODE.....	30
APPENDIX E—CATEGORY ANALYSIS CODE	31
APPENDIX F—CONTINUOUS ANALYSIS CODE.....	39

TABLE OF FIGURES

Figure 1. Percentage of small stream km in the Suwannee District that exceed a certain threshold.	4
---	----------

INTRODUCTION

The probabilistic design of the Status Network utilizes an unbiased, rigorous data set for the purpose of answering water-resource questions with mathematical confidence. The Network allows DEP to ask specific questions regarding water quality and answer those questions within statistical confidence limits. The design is planned with specific questions in mind.

Questions addressed by the Status Network monitoring design comprise three different scales: (1) the state of Florida as a whole, (2) regions of the state, and (3) large drainage basins, or drainage basin complexes (i.e., reporting units). In 2009, the Network was augmented to answer questions on a statewide and a zone (regional) basis with fewer sampling events. The questions that pertain to the Status Network relate to water quality *on a statewide and regional basis*; they are *not waterbody specific*. The Network is not designed to address the questions related to small drainage basins, counties, or individual bodies of water. The Integrated Water Resource Monitoring Program¹ (Copeland, et al., 1999) has allocated questions for these smaller areas be addressed by other monitoring. The design of the Status Network is for addressing statewide and coarse-scale questions with a high degree of statistical confidence.

Questions asked by the Status Network can be specific, such as whether the designated use of the water is being met. Examples of such questions include, but are not limited to,

- ☐ What percentage of small stream kilometers in the Suwannee basin has greater than 0.08 mg/L of phosphorus?
- ☐ What percentage of river miles exceeds standards for fecal coliform?
- ☐ What percentage of small lakes exceeds nitrate concentrations of 1.0 mg/L?
- ☐ Has the quality of water statewide improved since the last reporting period?
- ☐ What percentage of large lake hectares (or acres) exceeds standards for chlorophyll a?
- ☐ Do large lakes demonstrate worse Trophic State Indices than the last reporting period?

Status Network statements are easily represented and compared over time. Pie charts are used to summarize results. Figure 1 illustrates the first question (example small streams code seen in Appendix A). The light gray portion of diagram indicates 68% of stream lengths have dissolved oxygen above 5 mg/L (i.e., above the attainment level); blue shows that 32% of stream km are below 5 mg/L (not attaining). Therefore, of the 27135.165 km of streams throughout the state, 8683.253 km have concentrations below the dissolved oxygen threshold. Likewise, pie charts can be used to provide simple, easily understood illustrations of other questions.

The Status Network design provides advantages not offered by other designs. For one, the statistical confidence provided by the methodologies and random sampling design provide the advantage of knowing the proportion of non-attainment of an analyte within confidence bounds. Secondly, the random design generates proportion results that not only lend themselves to spatial comparisons (e.g., between one part of the state to another) but also temporal comparisons. Therefore, an objective statement with statistical confidence can be generated that tracks progress of the state as long as the network exists. Hence, the Network will have a unique ability

¹ The Integrated Water Resource Monitoring Program is sometimes referred to by the acronym, "IWRM."
<http://www.dep.state.fl.us/water/monitoring/docs/monitoring-strategy-2009.pdf>

to provide long-term tracking of statewide water quality, with statistical confidence, for decades to come. Comparisons can be made for the entire state through time, for an individual basin, or between different basins given enough sampling events have been conducted. In summary, the Status Network allows us to ask the questions that both the Department of Environmental Protection, and the Florida public, regularly ask. Simply stated: are conditions getting better, remaining the same, or getting worse?

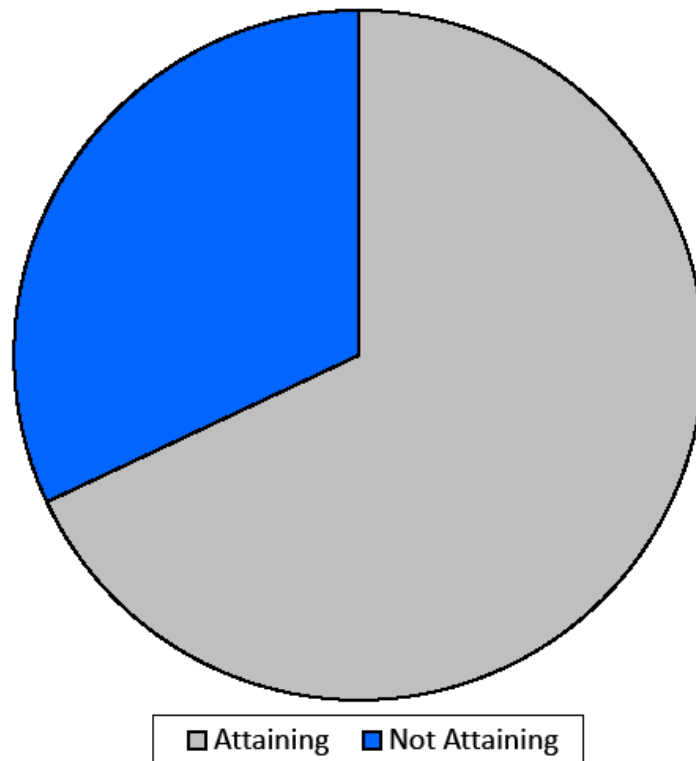


Figure 1. Percentage of small stream km that do not attain a certain threshold.

The document is divided into three sections. Part I will outline the operation of the Status Network and questions that may be addressed. Part II will describe quality assurance guidelines prior to data analysis. Part III takes an example resource, small lakes of the Zone 2 (Suwannee River District), to explain how the computer code used by the Status Network works.

PART I—THE STATUS NETWORK

The Status Network has three components. First, standardized protocols are employed in data acquisition, e.g.using standardized sample collection methods to minimize error associated with sampling. Second, the data population must be characterized. This means that the population of resources from which the data were collected must be characterized to statistically describe the variability of the distributions of the indicators sampled. Third, the calculated distributions are used to used to make inferences on the overall condition of the resources (i.e., the original, overall population that was sampled).

Questions addressed by the Status Network

Each year the analyses are conducted on the data on a statewide basis. After a minimum of three years the data will be collapsed and the analyses will be performed on the data in each of size zones. For the state and each zone a number of statistics will be calculated. These are performed through the extent and continuous variable estimate calculations. Appendix A gives example code for small streams. The following types of statistics can be assessed:

- ☐ What is the accessible proportion of a resource (surface or ground water)?
- ☐ How many samples were collected for each resource?
- ☐ What is the size of the resource in the zone?
- ☐ What proportion of the resources is dry?
- ☐ What proportion of surface water (i.e., large lakes, large rivers, small lakes, small streams) are exceeding standards?
- ☐ What proportion of ground water (i.e., as confined and unconfined aquifers) are exceeding standards?
- ☐ For surface water, what proportion of dissolved oxygen, fecal coliform, pH, unionized ammonia, and tropic state indices (or chlorophyll) are exceeding standards?
- ☐ For ground water, what proportion of arsenic, cadmium, chromium, lead, nitrate-nitrite, sodium, fluoride, and total coliform are exceeding standards?

PART II—DATA QUALITY ASSESSMENTS

Status Network data are acquired with Quality Assurance Protocols to insure integrity. Proper field and laboratory protocols are followed prior to their incorporation into the database. Data received from the database is then examined by the data analyst. This includes scanning data for outliers and erroneous values. The following are considerations in data quality:

Errors in the Data

The data should be screened for anomalies, outliers and questionable results based on proper ranges of values. Data that are orders of magnitude in variance from the central tendency are the most easily identified. Other values may fall outside of the logical range of values (e.g., negative values for NO₃). Continuous variable data (i.e., those not including raw count values) should have values greater than zero. Unusual values may warrant further investigation, including examination of original field records.

Outliers

One of the most common types of data errors are outliers. Outliers should not be discarded or arbitrarily defined. Though arbitrary standards can be set, all data handling is ultimately a matter for best professional judgment. For example, one way of defining outliers is identification of points beyond what are referred to as the upper fence. In order to determine the upper fence, the Inter Quartile Range (IQR) (the value of the upper quartile minus the value of the lower quartile) is determined. The upper fence is then found by adding the quantity 1.5 times the IQR to the upper quartile. However, a point 1.5 times the upper quartile is often a valid data point; on the

opposite end, erroneous data values may reside within this range. Ultimately, all decisions are subject to the analyst's best judgment given the data that is provided. Since Status Network analyses are nonparametric, being based on the rank of the data as opposed to being based on the data, outliers are less of a problem than they would constitute for parametric analyses. Removal of data from the dataset should only occur under the most obvious of circumstances. These include measurements that are mathematical impossibilities (e.g., pH values greater than 15), conditions that reflect physical impossibilities of condition (e.g., temperatures of 2700 C), mislabeled data, and laboratory instrumentation errors

Missing Values

Status Network sampling requires 10 collection points per zone for each resource. When less than 10 values are present in a specific zone, the reasons should be assigned to a write-up.

Detection Limits

Some reported data are at detection limits. The Status Network employs the reported value (MDL) for incorporation into CDFs.

Qualifier Codes

Status Network qualifier codes are recorded in Appendix B. Codes represent a range of problems. The type of qualifier, the reason for the particular qualifier, and the overall number of qualifiers should all be considered. A single qualifier can represent a variety of problems ranging from unimportant to serious. Additionally, the number of qualifiers may pose an additional problem; greater numbers are likely to have an influence on the CDFs. Yet, since the network design requires CDFs to be constructed from 60 values, removing qualified data is not recommended. Because a CDF is a function representing a nonparametric statistical distribution of underlying conditions, a single (or small number of) qualified data points will likely not affect the results. However, since both Statewide Report and 305b results report number of exceedances, qualified data values at or near important thresholds warrant further consideration. It is recommended that all qualified data remain in the analysis and that clear documentation of the reasons for the qualified data be presented alongside of the results. It is clear that CDFs constructed from heavily qualified data—or much data qualified near threshold ranges—will have compromised accuracy. Qualified values exceeding thresholds pose the most important questions.

Predetermined, arbitrary guidelines for how to handle various combinations of data qualifiers can be assigned. For example, pie charts could be flagged when the data includes 5 or more qualifiers. Whether or not such guidelines are established, data analysts and readers/editors of the reports must ultimately make decisions on a case-by-case basis as to whether or not to accept the findings. In any case, readers should be given enough information to make informed decisions about the resource(s) in question. Appendix B lists all the field and laboratory qualifiers; currently all are used in analyses.

Specific issues regarding data qualifiers include the following:

- (1) Samples held beyond holding times—bacteria data from the lab is submitted only when it is within holding times, or one day outside holding times. All data is currently used based on an

in-house study of the Environmental Assessment Section of the lab showing little degradation of samples up to one day outside holding times.

- (2) F and J qualified data—F and J qualified data may have failed QA and QC protocols, whether in the field or the lab.
- (3) W qualified data—for wells with metal casings, trace metal results may be compromised. Aspects of well construction may influence data values.
- (4) V qualified data—data that registers detection in both samples and blanks.

PART III—ANALYSIS PROCEDURES

The following is an example of data analysis on Status Network small lakes data. The script is an S-PLUS[®] code written by Tony Olsen, a US EPA statistician stationed at the US EPA facility in Corvallis, OR.. Analyses on other resources use similar script that has been adapted to the resource in question.

The following procedures have explanatory texts listed in *italics*; one time, or special information, is listed within gray text boxes.

Data is initially exported from the Oracle[®] database by Jay Silvanima, data manager for WMS.

Running the program in S-PLUS[®]

The following code runs in S-PLUS 6.0 Professional Release 2, S-PLUS 6.1 and 6.2 Professional Editions. Currently FDEP has the licences for S-Plus 7.2 and 8.0.

Start S-PLUS[®]

If the "psurvey.analysis" library folder is not installed, the following one-time installation procedure must be followed.

Acquire zipped library file from internet...

1. <http://www.epa.gov/nheerl/arm/analysispages/techinfoanalysis.htm>
2. psurvey.analysis 2.2 was used for these procedures but is updated over time. The current library is 2.6. The library versions may also change as S-PLUS versions are updated to 7.0; changes must be noted.
3. After downloading library, unzip and install in the Library folder in S-PLUS.
4. To find the location of the Library folder for installation, on the S-PLUS Menu find "Window," "Commands Window" and at the cursor type "getenv ('\$HOME')." This will give the location of S-PLUS on the HD. The "psurvey.analysis" folder must be placed directly within the library.
5. The second embedded folder must be used. Copy this one into the S-PLUS library. [Note: the psurvey.analysis folder that is to be installed contains ".Data" and ".Prefs" subfolders (and possibly others,

- depending on version).] Unless the folder is properly named, S-PLUS will not be able to call it. The name must be "psurvey.analysis."
6. Check for the proper installation of the library using Help...Available Help (on drop-down menu). When the library is installed it will be listed under the first option of the help drop-down list -- directly under "S-PLUS Help." This step is important since there is no error message to warn the user if the psurvey.analysis library failed to install.
 7. There is a folder on SOL Z for (1) libraries, (2) original scripts written by Tony Olsen for each individual resource (often written in "R" rather than "S-PLUS"), and (3) updated code that is S-PLUS compatible. These scripts will be assigned with the proper working library within ordered folders.

Select File....

Load library...

1. Select library "psurvey.analysis"
2. Check to make sure library installed under "Help...Available Help..."

Select File...

Chapter...

New Working Chapter...

(Puts data in folder at set location, e.g., D:\Program files\insightful
splus\users\doran\...)

Chapter folders can be labeled by resource (e.g., Small Lakes, Groundwater-
CA, Small Streams, etc.)

Open the Script

Select Open...

Go to the folder containing the program: "FLLakesAnalysis." (FLLakesAnalysis is Tony Olsen's original script for Florida small lakes of the Suwannee River District and is located in Appendix A).

Future Scripts will be stored on SOL Z by resource and S-PLUS version.

This step is useful for organizing the S-PLUS workspace but is not necessary to the operation of the program or proper output results. It allows error messages to be displayed below the script window and sends Report results to a separate window.

Options...

Text output routing...

1. Window for Normal Text Output (left pane)
Select "Report"
2. Window for Errors (right pane)
Select Default

Next load the data files. Similar to the installation of libraries, data files are initially loaded by one set of procedures but later saved within S-PLUS to be called by other commands. Both are listed below, with initial installation highlighted in gray.

For initial installation of data select,
File...

Import data...

From file...

1. Select two files to be imported into S-PLUS, one for site information and the other for data (e.g., in this case "siteinfo" and "SRSL0403"). The format of a "siteinfo" file can be seen in Appendix C.
2. Proper installation of the data files will include column headers on the second row having column names shaded gray. If this is not the case (i.e., column names have generic names like "Col 1") then actual first rows of data columns will include column names -- this will not run properly. [Note: This is a MS Excel-specific problem; pipe-delimited or comma delimited (*.csv) files appear to avoid the problems addressed below].
3. At this point you may have to check that the column headers will be properly installed. Select the second tab named "Options" and for the first choice on the upper left change the "Col names row" value to "1" (default is Auto); go four spaces down and make sure "Start row" has a value of "2" (instead of default of 1).
4. Check all numeric columns of the data and "siteinfo" files by holding the mouse cursor (downward pointing black arrow) over the row number. For example, column 3 in "siteinfo" is "Nest1.Wt." The label the appears when the cursor is held over number 3 should read "double" (for other proper column formats for "siteinfo" see Appendix C). If column 3 is not double,
 - a. Right click while the cursor is over number 3
 - b. On the menu that appears, go about half-way down to "Change Data Type..."
 - c. In the box that appears, go to the lower right corner and select "double" from the drop-down list.
 - d. Repeat this for all number containing rows (i.e., they should not be "factor" if they are numeric values or S-PLUS cannot make the calculations).
 - e. In the data file, SRSL0403, it may be easiest to start with the first data value (X10), hold down the left mouse button over the column number for variable X10 (Column 5) and drag the mouse to the right until the end of the data series is reached. Once these are all highlighted change the data types to "double" (repeat steps for "b" and following). Although this will change character values to double, we do not need them for this exercise.

If data has already been installed then it is stored within S-PLUS and can be acquired through the following steps. On the drop-down menu select

Data

Select Data...

On the upper right, under "Existing Data," choose the file names from the drop-down list: in this case "siteinfo" and "SRSL0403."

Running the script—Highlight relevant portions of text code in the Script window (code will be run in sections through the program...)

1. Hit run button (small black triangle near upper left of screen)
2. Session window will show output, while the bottom window will log errors
3. Run sections of code step-by-step in order to make sure each step runs properly (e.g., no errors). Sections of code are divided by comment lines (in S-PLUS these are specified by # symbols preceding text and will not execute with other script -- they are descriptive only).

Explanations for each portion of script:

The following illustrates the script and provides commentary. Portions of Script code are in text boxes (full code example for small streams is in Appendix A), and explanations are beneath in italics.

```
# Read in data from Florida using File: import data
names(siteinfo)
```

Read column headers into the Report window; this insures data are being read properly by the script.

```
# check units lat/long
plot(siteinfo$RANDOM.LONGITUDE, siteinfo$RANDOM.LATITUDE)
stem(siteinfo$RANDOM.LONGITUDE)
```

Whatever is happening with random latitude and longitude data are displayed graphically here. If degrees, minutes and seconds are displayed then the units have to be converted to decimal degrees; this is therefore a check.

```
# convert to Decimal degrees and do map projection
deg <- floor(siteinfo$RANDOM.LATITUDE/10000)
min <- floor((siteinfo$RANDOM.LATITUDE - deg*10000)/100)
sec <- siteinfo$RANDOM.LATITUDE - deg*10000 - min*100
siteinfo$latdd <- deg + min/60 + sec/3600
deg <- floor(siteinfo$RANDOM.LONGITUDE/10000)
min <- floor((siteinfo$RANDOM.LONGITUDE - deg*10000)/100)
sec <- siteinfo$RANDOM.LONGITUDE - deg*10000 - min*100
siteinfo$londd <- deg + min/60 + sec/3600
```

Decimal degree coordinates are created from degrees, minutes and seconds. Numeric conversions are made and then added together to give final "latdd" and "longdd" values.

```
# do equal area projection for variance estimation
tmp <- marinus(siteinfo$latdd, siteinfo$longdd)
siteinfo$xcoord <- tmp[, 'x']
siteinfo$ycoord <- tmp[, 'y']
```

In order to calculate variance, an equal area map projection must be created (marinus). This creates a map projection similar to ArcMap and is a common map projection. Creates a matrix, x and y, and assigns this to the siteinfo data frame.

```
# check design variables
table(siteinfo$CAN.BE.SAMPLED)
table(siteinfo$EXCLUSION.CATEGORY)
table(siteinfo$EXCLUSION.CRITERIA)
table(siteinfo$EXCLUSION.CATEGORY, siteinfo$CAN.BE.SAMPLED)

# create status and TNT variables
siteinfo$EXCLUSION.CATEGORY <- as.character(siteinfo$EXCLUSION.CATEGORY)
siteinfo$EXCLUSION.CATEGORY[siteinfo$EXCLUSION.CATEGORY == 'NA'] <- 'SAMPLED'
siteinfo$EXCLUSION.CATEGORY <- as.factor(siteinfo$EXCLUSION.CATEGORY)
levels(siteinfo$EXCLUSION.CATEGORY)
siteinfo$TNT <- siteinfo$EXCLUSION.CATEGORY
levels(siteinfo$TNT) <- list(T=c('SAMPLED', 'DENIED ACCESS', 'UNABLE TO OBTAIN
ACCESS'),
                             NT=c('DRY', 'WRONG RESOURCE/NOT PART OF TARGET POPULATION'))

table(siteinfo$EXCLUSION.CATEGORY, siteinfo$TNT)
```

Rather than scanning through a data frame, it is easier to look at summary information. This allows familiarity with the data.

TNT represents Target and Non-target samples. The exclusion categories are given but this is an additional simplification. The exclusion categories are subsumed into two categories: T and NT. The question here becomes which of the exclusions are targets? If someone were denied access then the presumption is you believe it was target, thus it is categorized under "T". Initially there was enough information to warrant visitation to the site -- given permission it would have been sampled.

```
##### adjust weights for use of over sample sites
### Size of sample frame 1400 lakes in Suwannee Basin
# adjust the weights
nr <- nrow(siteinfo)
siteinfo$wgt <- adjwgt.fcn(rep(TRUE,nr),
                           siteinfo$NEST1.WT,
                           siteinfo$REPORTING.UNIT,
                           framesize=c('SRWMD-1'=1400)
                           )
# following is temporary fix
siteinfo$wgt <- as.vector(siteinfo$wgt)
```

The adjust weights step accounts for oversampling. In a perfect world, a selection of 50 lakes would result in 50 samples: complete accessibility. All sums would equal 1400 if all sampling attempts were perfectly successful. But what if 50 samples required 70 attempts? Sample size

changes (by 70/50). This is the ratio adjustment to account for oversamples. [Entering "adjwgt" in the SPLUS command window will display the function description, see Appendix D].

The variable NEST1.WT is read into adjwgt.fcn from siteinfo. These are the original weights from siteinfo which were calculated from the original listframe (in this case, 1400 Suwannee District small lakes). NEST1 is the weight for a lake -- total lakes 1400/sample size for design value of 30 gives a weight of 46.67). The original GIS coverage had 1400 lakes and gives the listframe number used for the calculation. This is sent out with the documentation for the design. Analysis of a different basin will require changing the listframe number here, as well as data columns names, in order for the code to work. Otherwise, calculations of various resources employs nearly identical code.

```
##### Example Extent estimation

sites <- data.frame(siteID=1:nr, Use=rep(TRUE,nr) )
subpop <- data.frame(siteID=1:nr,
  Basin=rep('Suwannee Basin',nr) )
dsgn <- data.frame(siteID=1:nr,
  wgt=siteinfo$wgt,
  xcoord=siteinfo$xcoord ,
  ycoord=siteinfo$ycoord ,
  stratum=rep('None',nr)
)
data.cat <- data.frame(siteID=1:nr,
  EXCLUSION.CATEGORY=siteinfo$EXCLUSION.CATEGORY,
  TNTStatus=siteinfo$TNT
)
software.program <- 'S-PLUS'
ExtentEst <- cat.analysis.fcn(sites, subpop, dsgn, data.cat,
  type.cat=c(EXCLUSION.CATEGORY='Status',TNTStatus='Status'),
  popsize=c(None=1400),
  conf=95
)

#write out or export results
write.table(ExtentEst,file='ExtentEst.csv',sep=', ')
```

There are different kinds of data: categorical and continuous. Examples of categorical data are exclusion categories, or whether something was sampled or not. The "cat.analysis" function is used for these types of data. This first requires setting up data in 4 different frames: sites, subpop, dsgn, and data.cat. The part of the population examined is represented by subpop. Subpop will be employed when recombining all the data and the state and can group populations into smaller units (e.g., Reporting Unit, District, etc.). [Entering "cat.analysis" in the SPLUS command window will display the function description, see Appendix E].

"Dsgn" gives survey design used: it takes sites, weights, and stratification (though no stratification used here). Though x and y coordinates are called here, ultimately these are not used in making estimates for CDFs -- no geographic information goes into a CDF. [Though geographic information is employed within confidence limits or variance calculations (a local neighborhood variance calculation is found in cat.analysis). "Data.cat" -- Give the data used in the analysis, which is either 1) an exclusion categorical variable, or 2) a TNT variable.

"Cat.analysis" -- size of population can be specified and is part of included documentation. Could be 1400 lakes, 1500 km of streams, etc. "ExtentEst"-- Provides an estimated extent. For

example, estimate the total number of dry lakes in Suwannee. If 10% of those sampled are dry, there would be 140 in a sample size of 1400 total lakes.

```
##### Do summaries for data
# import data
names(SRSL0403)
# Note that have blank, primary and bottom data as rows.
# hence need to subset before do analyses

# Keep Primary data only
primary <- SRSL0403[SRSL0403$SAMPLE.TYPE == 'PRIMARY',]
# merge with siteinfo using menu command to primary.m
```

Before completion of the process, file merging must happen before the final results are produced—In the example code, we must merge “siteinfo” and “primary.” This step is needed because the SAMPLE.TYPE column contains three types of data: BLANK, PRIMARY, and BOTTOM. For this analysis, the primary data are chosen to create a new file with only those data. This file will be called "primary.m."

Data...

Merge...

Select the files under the Data block in upper left, ...

1. Data Set 1 must be "siteinfo"
2. Data Set 2 must be "primary" (select from drop down list) [Note: order is important]
3. Below, select radio button for "Specified Cols" -- this will activate the bottom two drop down lists.
4. For "Columns in Set 1" choose PK.Random.Sample.Location
5. For "Columns in Set 2" choose Station.Name (this will key on the same information in both files).
6. Under "Results", for "Save In:" type: primary.m

```
##### Example continuous indicator estimation
nr <- nrow(primary.m)
sites <- data.frame(siteID=primary.m$PK.RANDOM.SAMPLE.LOCATION,
Use=rep(TRUE,nr) )
subpop <- data.frame(siteID=primary.m$PK.RANDOM.SAMPLE.LOCATION,
  Basin=rep('Suwannee',nr) )
dsgn <- data.frame(siteID=primary.m$PK.RANDOM.SAMPLE.LOCATION,
  wgt=primary.m$wgt,
  xcoord=primary.m$xcoord ,
  ycoord=primary.m$ycoord ,
  stratum=rep('None',nr)
)
data.cont <- data.frame(siteID=primary.m$PK.RANDOM.SAMPLE.LOCATION,
  primary.m[,c('X10', 'X76')])
)

ExampleEst <- cont.analysis.fcn(sites, subpop, dsgn, data.cont,
  conf=95
)

write.table(ExampleEst$CDF,file='ExampleEstCDF.csv',sep=',')
write.table(ExampleEst$Pct,file='ExampleEstPCT.csv',sep=',')
write.table(ExampleEst$Tot,file='ExampleEstTOT.csv',sep=',')
```

The last portion of the code creates percent, total, and CDF calculations and writes them to tables. "Cont.analysis" is for use with the continuous variables (e.g., pH, nitrate, etc). [Entering "cont.analysis" in the SPLUS command window will display the function description, see Appendix F]. CDFs can only be created using continuous data; count data such as bacteria can theoretically be used, however tied data may generate problems. The way this is set up is the same with the categorical: sites, subpop, design, etc. This is a high level function that says you want to make CDF estimates for a range of variables and subpopulations at the same time. The CDFs generated are estimated population CDFs. These are what the CDF would look like if all lakes in the listframe were sampled. Medians calculated on raw data, and 50% calculation of these CDFs, will not be exact (but close).

The Program generates the results files under 4 different names in the chapter folder created above. In this example they went to D:\Program files\insightful\splus\users\doran_n\Small Lakes). The names are:

*ExampleEstCDF,
ExampleEstTot,
ExampleEstPCT, and
ExtentEst.*

These are comma delimited text files that can be easily manipulated in Excel or opened in Object Explorer (icon near top of screen with a yellow box).

Plotting CDFs

One way to plot the results is to open the text files in Excel, separate the results based on analyte (e.g., X10, which is temperature), and save the data for CDFs as separate files. [Note: There is probably a better way to do this but this is simple and will work].

1. Open the file "ExampleEstCDF" and save as an Excel spreadsheet. Separate rows of data based on analytes (e.g., x10 as a temperature file, x76 as turbidity, etc.).
2. Once files are broken out by analyte, open S-PLUS and select File, Import data, from file.. and select the file (e.g., "lake temperature").
3. S-PLUS will provide the headings (if they were included on the Excel spreadsheet). Highlight the following 4 columns (holding down CTRL-key after first selection...) → Value, Estimate.P, LCB95Pct.P, and UCB95Pct.P.
4. Select Graph, 2D Plot, Prob Y, and Line Plot (x, y1, y2...)
5. when the plot is created, it can be altered by the following steps
 - a. Insert Title...(Insert, Titles, Main, replace text with your title)
 - b. Click Y axis (once)...(Insert, Titles, Axis, type "Percent of small lakes")
 - c. Double-click Y-axis, (under axis scale select Linear)
 - d. Double-click "Value" on bottom of x-axis, replace with "Temperature"
 - e. Double-click lower line of the three on the graph—this is the lower confidence bound; select the Line tab, Line Style, and choose a dashed line
 - f. Double-click upper line of the three on the graph—this is the upper confidence bound; select the Line tab, Line Style, and choose a dashed line

- g. Double-click middle line of the three on the graph—this is the CDF; select the Line tab, Line Style, and choose 2 for weight (to make it easier to see).
- h. Insert Legend

REFERENCES

Copeland, R., Upchurch, S., Summers, K., Janicki, P., Hansard, P., Paulic, P., Maddox, G., Silvanima, J., and Craig, P., 1999, Overview of the Florida Department of Environmental Protection's Integrated Water Resource Monitoring Efforts and the Design Plan of the Status Network: Florida Department of Environmental Protection, Ambient Monitoring Section, 41 p.

APPENDIX A—EXAMPLE SMALL STREAMS CODE

```
# Purpose: Illustrate estimation for Florida small streams using SPlus
# Programmer: Tony Olsen and Chris Sedlacek
# (Combined Statewide assessment 2010)

# Load psurvey.analysis library using menu
# Packages: Load Packages click on psurvey.analysis in window

# Read in data from Florida using File menu
SS.SITES<-X2010SS.EXCLUSION
names(SS.SITES)

# convert to Decimal degrees and do map projection
deg <- floor(SS.SITES$RANDOM.LATITUDE/10000)
min <- floor((SS.SITES$RANDOM.LATITUDE - deg*10000)/100)
sec <- SS.SITES$RANDOM.LATITUDE - deg*10000 - min*100
SS.SITES$latdd <- deg + min/60 + sec/3600
deg <- floor(SS.SITES$RANDOM.LONGITUDE/10000)
min <- floor((SS.SITES$RANDOM.LONGITUDE - deg*10000)/100)
sec <- SS.SITES$RANDOM.LONGITUDE - deg*10000 - min*100
SS.SITES$londd <- deg + min/60 + sec/3600

# do equal area projection for variance estimation
tmp <- marinus(SS.SITES$latdd, SS.SITES$londd)
SS.SITES$xmarinus <- tmp[, 'x']
SS.SITES$ymarinus <- tmp[, 'y']

# check design variables
table(SS.SITES$CAN.BE.SAMPLED)
table(SS.SITES$EXCLUSION.CATEGORY)
table(SS.SITES$EXCLUSION.CRITERIA)
table(SS.SITES$EXCLUSION.CATEGORY, SS.SITES$CAN.BE.SAMPLED)
table(SS.SITES$EXCLUSION.CRITERIA, SS.SITES$EXCLUSION.CATEGORY)

# create status and TNT variables
SS.SITES$EXCLUSION.CATEGORY <- as.character(SS.SITES$EXCLUSION.CATEGORY)
SS.SITES$EXCLUSION.CATEGORY[SS.SITES$CAN.BE.SAMPLED == 'Y'] <- 'SAMPLED'
SS.SITES$EXCLUSION.CATEGORY <- as.factor(SS.SITES$EXCLUSION.CATEGORY)
levels(SS.SITES$EXCLUSION.CATEGORY)
SS.SITES$STATUS <- SS.SITES$EXCLUSION.CATEGORY
```

```

levels(SS.SITES$STATUS) <- list(T=c('SAMPLED', 'NO PERMISSION FROM OWNER', 'UNABLE TO ACCESS', 'OTHERWISE
UNSAMPLEABLE', 'OTHERWISE UNSAMPELABLE', 'DRY'),
                                NT=c('WRONG RESOURCE/NOT PART OF TARGET POPULATION') )

table(SS.SITES$EXCLUSION.CATEGORY, SS.SITES$STATUS)
summary(SS.SITES$STATUS)

##### adjust weights for use of over sample sites
# use only small stream sites that were evaluated: drop non-evaluated sites

# create basin variable
SS.SITES$basin <- SS.SITES$REPORTING.UNIT
table(SS.SITES$basin)

framesize <- c("ZONE 1"=13303.02,"ZONE 2"=2539.166,
               "ZONE 3"=4803.416,"ZONE 4"=4576.432,"ZONE 5"=1565.119,
               "ZONE 6"=348.012

               ) ### change to match combined resource stratum (I.E.-ADD THE TOTAL RIVER AND STREAM MILES
TOGETHER)

nr <- nrow(SS.SITES)
SS.SITES$wgt <- adjwgt(!is.nan(SS.SITES$STATUS),
                      SS.SITES$NEST1.WT,
                      SS.SITES$basin,
                      framesize=framesize
                      )

tapply(SS.SITES$wgt, SS.SITES$basin, sum)

### 2539.166 is the number of small stream kilometers in Zone 2 (Suwannee River Water ### Management District) stratum
##### Estimate stream km in population and by status category

sites <- data.frame(siteID=SS.SITES$PK.RANDOM.SAMPLE.LOCATION,
                   Use=rep(TRUE,nr))
subpop <- data.frame(siteID=SS.SITES$PK.RANDOM.SAMPLE.LOCATION,
                   Combined=rep("All Basins",nr),
                   basin=SS.SITES$basin )
dsgn <- data.frame(siteID=SS.SITES$PK.RANDOM.SAMPLE.LOCATION,
                  wgt=SS.SITES$wgt,
                  xcoord=SS.SITES$xmarinus ,
                  ycoord=SS.SITES$ymarinus ,
                  stratum=SS.SITES$basin
                  )

```

```

data.cat <- data.frame(siteID=SS.SITES$PK.RANDOM.SAMPLE.LOCATION,
                      EXCLUSION.CATEGORY=SS.SITES$EXCLUSION.CATEGORY,
                      TNTStatus=SS.SITES$STATUS)
software.program<-'S-PLUS'

ExtentEst <- cat.analysis(sites, subpop, dsgn, data.cat,
                        popsize=list(Combined=list("All Basins"=framesize),
                                      basin=list("ZONE 1"=framesize,"ZONE 2"=framesize,
                                                  "ZONE 3"=framesize,"ZONE 4"=framesize,"ZONE 5"=framesize,
                                                  "ZONE 6"=framesize
                                                  )),
                        conf=95
                        )

###write out or export results
write.table(ExtentEst,file='ExtentEst.csv',sep=',')

##### Do summaries for data
# import data
SS<-X2009SScomba
names(SS)

# look at sample type
summary(SS$Sample.Type)

# Note that have BLANK, BOTTOM and PRIMARY data in original spreadsheet.
# hence need to subset before do analyses
keep <- SS$Sample.Type == 'PRIMARY'

# merge with status
comb <- merge(SS.SITES, SS[keep,], by.x='PK.RANDOM.SAMPLE.LOCATION',
              by.y='Station.Name')

# check that have only PRIMARY data
summary(comb$SAMPLE.TYPE) # yes!

#####UIA Calculation#####

## Calculates unionized ammonia from total [NH4+NH3]

## Does not make ionic strength correction yet...

## Script requires temperature in degC, pH, and total ammonia value, in mg/L
## Warning! be sure to check odd UIA values, to see that required input variables exist
## Also, values are taken AS reported. If total ammonia/ammonium is below detection, so will be the UIA!!!

```

```

## get temperature vector, convert to degK
vTemp<-comb$X10
vTempK<-vTemp+273.2

## calculate equilibrium constant at sample Temperature
vEQconstant<-2729.92/vTempK + 0.0901821

## get UI fraction of total
vExponent<-vEQconstant - comb$X406
vDenom<-10^vExponent+1
vUIAfraction<-1/vDenom

## multiply fraction times total for UIA in lab units
vUIAraw<-vUIAfraction*comb$X610

## now convert to criterion units and write to file
comb$UIA<-vUIAraw*1.2158

#####End UIA Calculation#####

##### Example continuous indicator estimation
nr <- nrow(comb)
sites <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION, Use=rep(TRUE,nr) )
subpop <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                     Basin=rep('SRWMD-1',nr) )
dsgn <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                  wgt=comb$wgt,
                  xcoord=comb$xmarinus ,
                  ycoord=comb$ymarinus ,
                  stratum=comb$basin
                  )

data.cont <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,

                      comb[,c('Oxygen..Dissolved..Field','pH..Field','Coliform..Fecal..MF.','Chlorophyll.A..Monochromatic.','UIA')])

ExampleEst <- cont.analysis(sites, subpop, dsgn, data.cont)

# write out the results
write.table(ExampleEst$CDF,file='ExampleEstCDF.csv',sep=',')
write.table(ExampleEst$Pct,file='ExampleEstPCT.csv',sep=',')
write.table(ExampleEst$Tot,file='ExampleEstTOT.csv',sep=',')

```

```
#####2006 305b Pie Charts (Streams)#####
```

```
###
```

```
### Script to do categories for pie charts31616FecalColiform  
X31616cat<-cut(comb$Coliform..Fecal..MF.,breaks=c(0,399,1000000), include.lowest=TRUE)
```

```
comb$X31616cat <- X31616cat  
comb$X31616cat <- as.factor(comb$X31616cat)
```

```
nr <- nrow(comb)  
sites <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION, Use=rep(TRUE,nr) )  
subpop <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,  
                     Basin=comb$basin )  
dsgn <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,  
                  wgt=comb$wgt,  
                  xcoord=comb$xmarinus ,  
                  ycoord=comb$ymarinus ,  
                  stratum=comb$basin  
                  )  
data.cat31616 <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,  
                           X31616category=comb$X31616cat  
                           )
```

```
CategoryEst31616 <- cat.analysis(sites, subpop, dsgn, data.cat31616,  
                                popsize=list("CombinedBasins"=list("All Basins"=framesize),  
                                Basin=list("ZONE 1"=framesize, "ZONE 2"=framesize,  
                                "ZONE 3"=framesize, "ZONE 4"=framesize,  
                                "ZONE 5"=framesize, "ZONE 6"=framesize)  
                                )  
                                )
```

```
# write out the results  
write.table(CategoryEst31616,file='CategoryExample31616FecalColiform.csv',sep=',')
```

```
###
```

```
### Script to do categories for pie charts299  
X299cat<-cut(comb$Oxygen..Dissolved..Field,breaks=c(0,4.999,1000000), include.lowest=TRUE)
```

```
comb$X299cat <- X299cat  
comb$X299cat <- as.factor(comb$X299cat)
```

```
nr <- nrow(comb)  
sites <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION, Use=rep(TRUE,nr) )  
subpop <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,  
                     Basin=comb$basin )
```

```

dsgn <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                  wgt=comb$wgt,
                  xcoord=comb$xmarinus ,
                  ycoord=comb$ymarinus ,
                  stratum=comb$basin
                  )
data.cat299 <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                        X299category=comb$X299cat
                        )

CategoryEst299 <- cat.analysis(sites, subpop, dsgn, data.cat299,
                             popsize=list("CombinedBasins"=list("All Basins"=framesize),
                             Basin=list("ZONE 1"=framesize, "ZONE 2"=framesize,
                                         "ZONE 3"=framesize, "ZONE 4"=framesize,
                                         "ZONE 5"=framesize, "ZONE 6"=framesize)
                             )
                             )

# write out the results
write.table(CategoryEst299,file='CategoryExample299D0.csv',sep=',')

###_____

### Script to do categories for pie charts406
X406cat <- cut(comb$pH..Field, breaks=c(0,5.999,8.5,14), include.lowest=TRUE)
comb$X406cat <- X406cat
comb$X406cat <- as.factor(comb$X406cat)

nr <- nrow(comb)
sites <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION, Use=rep(TRUE,nr) )
subpop <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                    Basin=comb$basin )
dsgn <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                  wgt=comb$wgt,
                  xcoord=comb$xmarinus ,
                  ycoord=comb$ymarinus ,
                  stratum=comb$basin
                  )
data.cat406 <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                        X406category=comb$X406cat
                        )

CategoryEst406 <- cat.analysis(sites, subpop, dsgn, data.cat406,
                             popsize=list("CombinedBasins"=list("All Basins"=framesize),
                             Basin=list("ZONE 1"=framesize, "ZONE 2"=framesize,

```

```

        "ZONE 3"=framesize, "ZONE 4"=framesize,
        "ZONE 5"=framesize, "ZONE 6"=framesize)
    )
)

# write out the results
write.table(CategoryEst406,file='CategoryExample406pH.csv',sep=',')

###_____

#### Script to do categories for pie chartsUIA
UIAcat <- cut(comb$UIA, breaks=c(0,0.02,1000000), include.lowest=TRUE)
comb$UIAcat <- UIAcat
comb$UIAcat <- as.factor(comb$UIAcat)

nr <- nrow(comb)
sites <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION, Use=rep(TRUE,nr) )
subpop <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                     Basin=comb$basin )
dsgn <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                  wgt=comb$wgt,
                  xcoord=comb$xmarinus ,
                  ycoord=comb$ymarinus ,
                  stratum=comb$basin
                  )
data.catUIA <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                         UIAcategory=comb$UIAcat
                         )

CategoryEstUIA <- cat.analysis(sites, subpop, dsgn, data.catUIA,
                              popsize=list("CombinedBasins"=list("All Basins"=framesize),
                              Basin=list("ZONE 1"=framesize, "ZONE 2"=framesize,
                              "ZONE 3"=framesize, "ZONE 4"=framesize,
                              "ZONE 5"=framesize, "ZONE 6"=framesize)
                              )
)

# write out the results
write.table(CategoryEstUIA,file='CategoryExampleUIA.csv',sep=',')

###_____

#### Script to do categories for pie charts32211
X32211cat<-cut(comb$Chlorophyll.A..Monochromatic.,breaks=c(0,20,10000), include.lowest=TRUE)

```

```

comb$X32211cat <- X32211cat
comb$X32211cat <- as.factor(comb$X32211cat)

nr <- nrow(comb)
sites <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION, Use=rep(TRUE,nr) )
subpop <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                     Basin=comb$basin )
dsgn <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                  wgt=comb$wgt,
                  xcoord=comb$xmarinus ,
                  ycoord=comb$ymarinus ,
                  stratum=comb$basin
                  )
data.cat32211 <- data.frame(siteID=comb$PK.RANDOM.SAMPLE.LOCATION,
                          X32211category=comb$X32211cat
                          )

CategoryEst32211 <- cat.analysis(sites, subpop, dsgn, data.cat32211,
                                popsize=list("CombinedBasins"=list("All Basins"=framesize),
                                Basin=list("ZONE 1"=framesize, "ZONE 2"=framesize,
                                "ZONE 3"=framesize, "ZONE 4"=framesize,
                                "ZONE 5"=framesize, "ZONE 6"=framesize)
                                )
)

# write out the results
write.table(CategoryEst32211,file='CategoryExample32211Chlorophylla.csv',sep=',')

###_____

#####
#####VALUE QUALIFIER CODE#####
#####

## This script uses as input the transposed MSAccess project file.
## The script should be run "obviously" prior to converting the data types, as this will
## remove the data qualifiers.
## Note: results can have more than one qualifier. The sum of the qualifier occurrences will not equal the total
number of results.

## IF YOU ARE USING THIS SCRIPT FOR SOMETHING IMPORTANT, CHECK A FEW VALUES BEFORE STAKING YOUR REPUTATION ON THE
OUTPUT!!

## First read the file into a new data frame. Change the line below to read the correct file.
names(SRSS0404ReRun)

```

```

## Now get list of VQ columns and number
Xpars <- names(SRSS0404ReRun)[substring(names(SRSS0404ReRun),1,2) == 'VQ']
nXpar<-length(Xpars)

## Initialize VQ count holders
nvals<-NULL
NVQ<-NULL
UVQ<-NULL
IVQ<-NULL
QVQ<-NULL
LVQ<-NULL
BVQ<-NULL
OVQ<-NULL
KVQ<-NULL
VVQ<-NULL
JVQ<-NULL
FVQ<-NULL

## loop thru column name vector Xpar, and for each parameter, obtain counts
for (i in 1:nXpar)
{
  ## concatenate data frame and column name
  nthXpar<-eval(parse(text=paste("SRSS0404ReRun$",Xpars[i],sep="")))

  ## get number of values
  nvals<-c(nvals,length(nthXpar))

  ## get number of no qualified
  NVQ<-c(NVQ,length(which.na(nthXpar)))

  ## get number of U qualified
  UVQ<-c(UVQ,length(grep("U",substring(nthXpar,1,5))))

  ## get number of I qualified
  IVQ<-c(IVQ,length(grep("I",substring(nthXpar,1,5))))

  ## get number of K qualified
  KVQ<-c(KVQ,length(grep("K",substring(nthXpar,1,5))))

  ## get number of L qualified
  LVQ<-c(LVQ,length(grep("L",substring(nthXpar,1,5))))

  ## get number of V qualified
  VVQ<-c(VVQ,length(grep("V",substring(nthXpar,1,5))))
}

```

```

## get number of O qualified
OVQ<-c(OVQ,length(grep("O",substring(nthXpar,1,5))))

## get number of Q qualified
QVQ<-c(QVQ,length(grep("Q",substring(nthXpar,1,5))))

## get number of B qualified
BVQ<-c(BVQ,length(grep("B",substring(nthXpar,1,5))))

## get number of J qualified
JVQ<-c(JVQ,length(grep("J",substring(nthXpar,1,5))))

## get number of F qualified
FVQ<-c(FVQ,length(grep("F",substring(nthXpar,1,5))))
next
}

## make output table
xvec<-c(nvals,NVQ,UVQ,IVQ,KVQ,LVQ,VVQ,OVQ,QVQ,BVQ,JVQ,FVQ)
xmat<-matrix(xvec,length(xvec)/12,12)
dimnames(xmat)<-list(Xpars,c("N","NONE","U","I","K","L","V","O","Q","B","J","F"))
write.table(xmat,file='VQTable.csv',sep=',')

```

```

#####
#####END QUALIFER CODE#####

```

APPENDIX B—DATA QUALIFIERS

Value Qualifiers [from 2008 QA Rule 62-160.700 Table 1 (Data Qualifier Codes)]

Qualifier	Description
U	Indicates that the compound was analyzed for but not detected. The reported value shall be the method detection limit.
A	Value reported is the average of two or more determinations.
B	Colony counts were outside acceptable range. The value reported is an estimated count (This code applies to microbiological tests and specifically to membrane filter colony counts.)
I	The reported value is greater than or equal to the laboratory method detection limit but less than the laboratory practical quantification limit.
T	Value reported is less than the laboratory method detection limit.
K	Off-scale low. The actual value is known to be less than the value given.
N	Presumptive evidence of presence of material; component tentatively identified based on mass spectral library search or there is an indication that the analyte is present, but quality control requirements for the confirmation were not met.
O	Sampled but analysis lost or not performed.
Q	Sample held beyond the accepted holding time.
L	Off-scale high. The actual value is known to be greater than the value given.
J	Estimate value. Shall be accompanied by a detailed explanation to justify the reason(s) for designating the value as estimated. Examples of situations in which a “J” code must be reported include: instances where a quality control item associated with the reported value failed to meet the established quality control criteria (the specific failure must be identified); instances when the sample matrix interfered with the ability to make any accurate determination; instances when data are questionable because of improper laboratory or field protocols (e.g., composite sample was collected instead of a grab sample); instances when the analyte was detected at or above the method detection limit in a blank other than the method blank (such as calibration blank or field-generated blanks and the value of 10 times the blank value was equal to or greater than the associated sample value); or instances when the field or laboratory calibrations or calibration verifications did not meet calibration acceptance criteria.
V	Indicates that the analyte was detected at or above the method detection limit in both the sample and the associated method blank and the value of 10 times the blank value was equal to or greater than the associated sample value.
X	Indicates, when reporting results from a Stream Condition Index Analysis (LT 7200 and FS 7420), that insufficient individuals were present in the sample to achieve a minimum of 280 organisms for identification (the method calls for two aliquots of 140-160 organisms), suggesting either extreme environmental stress or a sampling error.
Y	The laboratory analysis was from an unpreserved or improperly preserved sample. The data may not be accurate.
Z	Too many colonies were present for accurate counting. Historically, this condition has been reported as “too numerous to count” (TNTC). The “Z” qualifier code shall be reported when the total number of colonies of all types is more than 200 in all dilutions of the sample. When applicable to the observed test results, a numeric value for the colony count for the microorganism tested shall be estimated from the highest dilution factor (smallest sample volume) used for the test and reported with the qualifier code
!	Indicates that the reported value deviates from historically established concentration ranges.
?	Data are rejected and should not be used. Some or all of the quality control data for the analyte were outside criteria, and the presence or absence of the analyte cannot be determined from the data.

Note: italicized descriptions deviate from EPA and/or DEP QAS descriptions.

Missing Values: blank

*Notes: The W qualifier is to be used in the following ways. 1) If turbidity is greater than 100 NTU, all analytes coming from that well will be qualified with a W. 2) If the well currently has, or historically had, a water level recording device employing a lead weight, all lead values coming from that well will be qualified with a W. 3) All VOC's will be qualified for each glued PVC well. 4) The following detections of analytes coming from galvanized steel wells will be qualified with a W: iron, manganese, zinc, cadmium. 5) The following detections of analytes coming from stainless steel wells will be qualified with a W: nickel, chromium. 6) All detections of trace metals coming from any type of iron well will be qualified with a W.

APPENDIX C—EXAMPLE SITEINFO FILE

<i>Column No.</i>	<i>Column Label</i>	<i>Correct Format</i>
1	Pk.random.sample.location	factor
2	Fk.epa.random	factor
3	Nest1.Wt	double
4	Epa.oversample	double
5	Epa.division	double
6	Resource.type	factor
7	Reporting.unit	factor
8	Random.latitude	double
9	Random.longitude	double
10	Hydrologic.unit.code	factor
11	Hydrologic.unit.name	factor
12	Can.be.sampled	factor
13	Exclusion.category	factor
14	Exclusion.criteria	factor
15	Sampled.date	factor
16	Analysis.suite	factor
17	Locational.datum	factor
18	Status*	factor
19	latdd*	double
20	longdd*	double
21	xmarinus*	double
22	ymarinus*	double
23	TNT*	factor
24	wgt*	double

*These columns were added by the FLLakesAnalysis execution rather than being part of the original data.

APPENDIX D—ADJUST WEIGHT CODE

```
#####  
# Function:  adjwgt  
# Programmer: Tony Olsen  
# Date:      February 25, 2004  
# Description:  
# Purpose of this function is to adjust initial survey design weights when  
# implementation results in use of oversample sites or when it is desired to  
# have final weights sum to known frame size. Adjusted weights are equal to  
# initial weight * framesize/sum(initial weights). The adjustment is done  
# separately for each category specified in wtcat.  
# Input:  
#   sites = the logical value for each site, where TRUE = include the site  
#           and FALSE = do not include the site.  
#   wgt = the initial weight (inverse of the sample inclusion probability)  
#         for each site.  
#   wtcat = the weight adjustment category name for each site.  
#   framesize = the known size of the frame for each category name in wtcat,  
#               which must have the names attribute set to match the category names  
#               used in wtcat.  
# Output:  
#   A vector of adjusted weights, where the adjusted weight is set to zero  
#   for sites that have the logical value in the sites argument set to  
#   FALSE.  
# Other Functions Required: None  
# Examples:  
#   sites <- as.logical(rep(rep(c("TRUE","FALSE"), c(9,1)), 5))  
#   wgt <- runif(50, 10, 100)  
#   wtcat <- rep(c("A","B"), c(30, 20))  
#   framesize <- c(15, 10)  
#   names(framesize) <- c("A","B")  
#   adjwgt(sites, wgt, wtcat, framesize)  
#####  
# Sum initial weights by wtcat for evaluated sites  
wgtsum <- tapply(wgt[sites], wtcat[sites], sum)  
# Adjustment factor to be applied to weights for adjustment category  
adjfac <- framesize/wgtsum[match(names(framesize), names(wgtsum))]  
wtadj <- adjfac[match(wtcat, names(adjfac))]  
adjwgt <- wgt * wtadj  
adjwgt[!sites] <- 0  
as.vector(adjwgt)
```

APPENDIX E—CATEGORY ANALYSIS CODE

```
#####  
# Function:    cat.analysis  
# Programmers: Tony Olsen  
#             Tom Kincaid  
# Date:       March 31, 2005  
# Description:  
#   This function organizes input and output for analysis of categorical data  
#   generated by a probability survey.  Input can be either an object of class  
#   psurvey.analysis (see the documentation for function psurvey.analysis)  
#   or through use of the other arguments to this function.  
#   Input:  
#     sites = a data frame consisting of two variables: the first variable is  
#             site IDs, and the second variable is a logical vector indicating which  
#             sites to use in the analysis.  If psurvey.obj is not provided, then  
#             this argument is required.  The default is NULL.  
#     subpop = a data frame describing sets of populations and subpopulations  
#             for which estimates will be calculated.  The first variable is site  
#             IDs.  Each subsequent variable identifies a Type of population, where  
#             the variable name is used to identify Type.  A Type variable  
#             identifies each site with one of the subpopulations of that Type.  If  
#             psurvey.obj is not provided, then this argument is required.  The  
#             default is NULL.  
#     design = a data frame consisting of design variables.  If psurvey.obj is  
#             not provided, then this argument is required.  The default is NULL.  
#             Variables should be named as follows:  
#               siteID = site IDs  
#               wgt = final adjusted weights, which are either the weights for a  
#                   single-stage sample or the stage two weights for a two-stage  
#                   sample  
#               xcoord = x-coordinates for location, which are either the x-  
#                   coordinates for a single-stage sample or the stage two x-  
#                   coordinates for a two-stage sample  
#               ycoord = y-coordinates for location, which are either the y-  
#                   coordinates for a single-stage sample or the stage two y-  
#                   coordinates for a two-stage sample  
#               stratum = the stratum codes  
#               cluster = the stage one sampling unit (primary sampling unit or  
#                   cluster) codes  
#               wgt1 = final adjusted stage one weights  
#               xcoord1 = the stage one x-coordinates for location  
#               ycoord1 = the stage one y-coordinates for location  
#     data.cat = a data frame of categorical response variables.  The first  
#               variable is site IDs.  Subsequent variables are response variables.
```

```

#       Missing data (NA) is allowed. If psurvey.obj is not provided, then
#       this argument is required. The default is NULL.
#       N.cluster = the number of stage one sampling units in the resource, which
#       is required for calculation of finite and continuous population
#       correction factors for a two-stage sample. For a stratified sample
#       this variable must be a vector containing a value for each stratum and
#       must have the names attribute set to identify the stratum codes. The
#       default is NULL.
#       popsize = the known size of the resource - the total number of sampling
#       units of a finite resource or the measure of an extensive resource,
#       which is required for calculation of finite and continuous population
#       correction factors for a single-stage sample. This argument is also
#       used to adjust estimators for the known size of a resource. The
#       argument must be in the form of a list containing an entry for each
#       population Type in the subpop data frame, where NULL is a valid entry
#       for a population Type. The list must be named using the variable
#       names for population Types in subpop. If a population Type doesn't
#       contain subpopulations, then the element of the list is either a
#       single value for an unstratified sample or a vector containing a value
#       for each stratum for a stratified sample, where the vector must have
#       the names attribute set to identify the stratum codes. If a
#       population Type contains subpopulations, then the element of the list
#       is a list containing an element for each subpopulation, where the list
#       is named using the subpopulation names. The element for each
#       subpopulation will be either a single value for an unstratified sample
#       or a named vector of values for a stratified sample. The default is
#       NULL.
#       stage1size = the known size of the stage one sampling units of a two-
#       stage sample, which is required for calculation of finite and
#       continuous population correction factors for a two-stage sample and
#       must have the names attribute set to identify the stage one sampling
#       unit codes. For a stratified sample, the names attribute must be set
#       to identify both stratum codes and stage one sampling unit codes using
#       a convention where the two codes are separated by the # symbol, e.g.,
#       "Stratum 1#Cluster 1". The default is NULL.
#       support = the support value for each site - the value one (1) for a
#       site from a finite resource or the measure of the sampling unit
#       associated with a site from an extensive resource, which is required
#       for calculation of finite and continuous population correction
#       factors. The default is NULL.
#       swgt = the size-weight for each site, which is the stage two size-weight
#       for a two-stage sample. The default is NULL.
#       swgt1 = the stage one size-weight for each site. The default is NULL.
#       unitsize = the known sum of the size-weights of the resource. The
#       argument must be in the form of a list containing an entry for each

```

```

#      population Type in the subpop data frame, where NULL is a valid entry
#      for a population Type. The list must be named using the variable
#      names for population Types in subpop. If a population Type doesn't
#      contain subpopulations, then the element of the list is either a
#      single value for an unstratified sample or a vector containing a value
#      for each stratum for a stratified sample, where the vector must have
#      the names attribute set to identify the stratum codes. If a
#      population Type contains subpopulations, then the element of the list
#      is a list containing an element for each subpopulation, where the list
#      is named using the subpopulation names. The element for each
#      subpopulation will be either a single value for an unstratified sample
#      or a named vector of values for a stratified sample. The default is
#      NULL.
#      vartype = the choice of variance estimator, where "Local" = local mean
#      estimator and "SRS" = SRS estimator. The default is "Local".
#      conf = the confidence level. The default is 95%.
#      psurvey.obj = A list of class psurvey.analysis that was produced by the
#      function psurvey.analysis. Depending on input to that function,
#      some elements of the list may be NULL. The default is NULL.
#      Output:
#      A data frame of population estimates for all combinations of population
#      Types, subpopulations within Types, response variables, and categories
#      within each response variable. Estimates are provided for proportion
#      and size of the population plus standard error and confidence interval
#      estimates.
#      Other Functions Required:
#      dframe.check - check site IDs, the sites data frame, the subpop data
#      frame, the data.cat data frame, and the data.cont data frame to assure
#      valid contents and, as necessary, create the sites data frame and the
#      subpop data frame
#      input.check - check input values for errors, consistency, and
#      compatibility with psurvey.analysis analytical functions
#      category.est - estimate proportion (expressed as percent) and size of a
#      resource in each of a set of categories
#      Examples:
#      mysiteID <- paste("Site", 1:100, sep="")
#      mysites <- data.frame(siteID=mysiteID, Active=rep(TRUE, 100))
#      mysubpop <- data.frame(siteID=mysiteID, Resource.Class=rep(c("A","B"),
#      c(55,45)))
#      mydesign <- data.frame(siteID=mysiteID, wgt=runif(100, 10, 100),
#      xcoord=runif(100), ycoord=runif(100), stratum=rep(c("Stratum1",
#      "Stratum2"), 50))
#      mydata.cat <- data.frame(siteID=mysiteID, CatVar=rep(c("north", "south",
#      "east", "west"), 25))
#      cat.analysis(sites=mysites, subpop=mysubpop, design=mydesign,

```

```

#       data.cat=mydata.cat)
#
#       mysites <- data.frame(siteID=mysiteID, Active=rep(c(TRUE, FALSE, TRUE,
#       TRUE), 25))
#       cat.analysis(sites=mysites, subpop=mysubpop, design=mydesign,
#       data.cat=mydata.cat)
#####
# Begin the section when an object of class "psurvey.analysis" was input to the
# function
if(inherits(psurvey.obj, "psurvey.analysis")) {
  # Assign variables from the input list
  sites <- psurvey.obj$sites
  subpop <- psurvey.obj$subpop
  design <- psurvey.obj$design
  data.cat <- psurvey.obj$data.cat
  N.cluster <- psurvey.obj$N.cluster
  popsize <- psurvey.obj$popsize
  stage1size <- psurvey.obj$stage1size
  support <- psurvey.obj$support
  swgt <- psurvey.obj$swgt
  swgt1 <- psurvey.obj$swgt1
  unitsize <- psurvey.obj$unitsize
  stratum.ind <- psurvey.obj$stratum.ind
  vartype <- psurvey.obj$vartype
  conf <- psurvey.obj$conf
}
else {
  # Check that the required data frames have been provided
  if(is.null(sites)) stop("\nThe sites data frame must be provided.")
  if(!is.data.frame(sites))
    stop("\nThe sites argument must be a data frame.")
  if(is.null(subpop))
    stop("\nThe subpop data frame must be provided.")
  if(!is.data.frame(subpop))
    stop("\nThe subpop argument must be a data frame.")
  if(is.null(design))
    stop("\nThe design data frame must be provided.")
  if(!is.data.frame(design))
    stop("\nThe design argument must be a data frame.")
  if(is.null(data.cat))
    stop("\nThe data.cat data frame must be provided.")
  if(!is.data.frame(data.cat))
    stop("\nThe data.cat argument must be a data frame.")
  # Check the design data frame for required names
  design.names <- names(design)

```

```

        temp <- match(design.names, c("siteID", "wgt", "xcoord", "ycoord", "stratum",
        "cluster", "wgt1",
        "xcoord1", "ycoord1"), nomatch = 0)
if(any(temp == 0))
    stop("\nThe names used in the design data frame do not match the required names.")
    # Check site IDs, the sites data frame, the subpop data frame, and the data.cat
# data frame to assure valid contents
    temp <- dframe.check(dim(design)[1], design$siteID, sites, subpop, data.cat, NULL,
design.names)
sites <- temp$sites
subpop <- temp$subpop
data.cat <- temp$data.cat
    # As necessary, remove rows of the design, sites, subpop, and data.cat data
# frames and elements of the support, swgt, and swgt1 vectors based on the
# logical variable in the sites data frame
subpop <- subpop[sites[, 2], ]
design <- design[sites[, 2], ]
data.cat <- data.cat[sites[, 2], ]
if(!is.null(support))
    support <- support[sites[, 2]]
if(!is.null(swgt))
    swgt <- swgt[sites[, 2]]
if(!is.null(swgt1))
    swgt1 <- swgt1[sites[, 2]]
sites <- sites[sites[, 2], ]
# Assign variables from the design data frame
siteID <- design$siteID
wgt <- design$wgt
xcoord <- design$xcoord
ycoord <- design$ycoord
stratum <- design$stratum
cluster <- design$cluster
wgt1 <- design$wgt1
xcoord1 <- design$xcoord1
ycoord1 <- design$ycoord1
# Determine whether the sample is stratified
stratum.ind <- length(unique(stratum)) > 1
    # If the sample is stratified, convert stratum to a factor, determine stratum
# levels, and calculate number of strata
if(stratum.ind) {
    stratum <- factor(stratum)
    stratum.levels <- levels(stratum)
    nstrata <- length(stratum.levels)
}
else {

```

```

        stratum.levels <- NULL
        nstrata <- NULL
    }
    # Determine whether the sample has two stages
    cluster.ind <- length(unique(cluster)) > 1
        # If the sample has two stages, convert cluster to a factor, determine cluster
    # levels, and calculate number of clusters
    if(cluster.ind) {
        if(stratum.ind) {
            cluster.in <- cluster
            cluster <- tapply(cluster, stratum, factor)
            cluster.levels <- sapply(cluster, levels, simplify = FALSE)
            ncluster <- sapply(cluster.levels, length)
        }
        else {
            cluster <- factor(cluster)
            cluster.levels <- levels(cluster)
            ncluster <- length(cluster.levels)
        }
    }
    # Determine whether the population correction factor is to be used
    pcfactor.ind <- !is.null(support)
    # Determine whether the sample uses size-weights
    swgt.ind <- !is.null(swgt)
    # Determine the number of response values
    nresp <- dim(design)[1]
    # Check for compatibility of input values
    temp <- input.check(nresp, wgt, NULL, NULL, xcoord, ycoord, stratum.ind, stratum, stratum.levels,
        nstrata, cluster.ind, cluster, cluster.levels, ncluster, N.cluster, wgt1, xcoord1, ycoord1,
        popsize, stage1size, pcfactor.ind, support, swgt.ind, swgt, swgt1, unitsize, vartype, conf,
        subpop = subpop)
    N.cluster <- temp$N.cluster
    popsize <- temp$popsize
    stage1size <- temp$stage1size
    unitsize <- temp$unitsize
    # If the sample was stratified and had two stages, then reset cluster to its
    # input value
    if(stratum.ind && cluster.ind) cluster <- cluster.in
    # As necessary, assign missing values to the design variables
    if(is.null(xcoord)) xcoord <- rep(NA, nresp)
    if(is.null(ycoord))
        ycoord <- rep(NA, nresp)
    if(is.null(stratum))
        stratum <- rep(NA, nresp)
    if(is.null(cluster))

```

```

        cluster <- rep(NA, nresp)
    if(is.null(wgt1))
        wgt1 <- rep(NA, nresp)
    if(is.null(xcoord1))
        xcoord1 <- rep(NA, nresp)
    if(is.null(ycoord1))
        ycoord1 <- rep(NA, nresp)
    # Recreate the design data frame
    design <- data.frame(siteID = siteID, wgt = wgt, xcoord = xcoord, ycoord = ycoord, stratum =
stratum,
                        cluster = cluster, wgt1 = wgt1, xcoord1 = xcoord1, ycoord1 = ycoord1)
}
# Loop through all response variables
nvar <- dim(data.cat)[2]
varnames <- names(data.cat)
ntypes <- dim(subpop)[2]
typenames <- names(subpop)
nrow <- 0
for(ivar in 2:nvar) {
    # Loop through all types of populations
    for(itype in 2:ntypes) {
        # Find unique subpopulations of this type of population
        subpopnames <- levels(factor(subpop[, itype]))
        # Loop through all subpopulations of this type
        for(isubpop in 1:length(subpopnames)) {
            # Select sites in a subpopulation
            subpop.ind <- subpop[, itype] == subpopnames[isubpop]
            # Determine whether the subpopulation is empty
            if(all(is.na(data.cat[subpop.ind, ivar]))) {
                warning(paste("Subpopulation", subpopnames[isubpop], "of
population type",
                            typenames[itype], "for indicator", varname[ivar],
                            "\ncontains no data.\n"))
            }
            next
        }
        # Determine whether the subpopulation contains a single value
        if(sum(!is.na(data.cat[subpop.ind, ivar])) == 1) {
            warning(paste("Subpopulation", subpopnames[isubpop], "of population
type",
                            typenames[itype], "for indicator", varname[ivar],
                            "\ncontains a single value. No analysis was performed.\n"))
        }
        next
    }
    # For a stratified sample, remove values from N.cluster, popsize, stage1size,
    # and unitsize for strata that do not occur in the subpopulation

```

```

        if(stratum.ind) {
            temp.N.cluster <- N.cluster[!is.na(match(names(N.cluster), unique(design[
                subpop.ind, 5])))]
            temp.popsiz <- popsize[[itype -
1]][[isubpop]][!is.na(match(names(popsiz[
                itype - 1]][[isubpop]]), unique(design[subpop.ind, 5])))]
            temp.stage1size <- stage1size[!is.na(match(names(stage1size),
unique(design[
                subpop.ind, 5])))]
            temp.unitsize <- unitsize[!is.na(match(names(unitsize), unique(design[
                subpop.ind, 5])))]
        }
        else {
            temp.N.cluster <- N.cluster
            temp.popsiz <- popsize[[itype - 1]][[isubpop]]
            temp.stage1size <- stage1size
            temp.unitsize <- unitsize
        }
        # Estimate category proportions for the response variable
        temp.cat <- category.est(catvar = data.cat[subpop.ind, ivar], wgt = design[
            subpop.ind, 2], x = design[subpop.ind, 3], y = design[subpop.ind, 4],
            stratum = design[subpop.ind, 5], cluster = design[subpop.ind, 6], N.cluster
            = temp.N.cluster, wgt1 = design[subpop.ind, 7], x1 = design[subpop.ind,
            8], y1 = design[subpop.ind, 9], popsize = temp.popsiz, stage1size =
            temp.stage1size, support = support[subpop.ind], swgt = swgt[subpop.ind],
            swgt1 = swgt1[subpop.ind], unitsize = temp.unitsize, vartype = vartype,
            conf = conf, check.ind = FALSE)
        # Assign category proportion estimates for the response variable to a data frame
        if(nrow == 0) {
            nn <- dim(temp.cat)[1]
            catsum <- data.frame(Type = rep(typhenames[itype], nn), Subpopulation = rep(
                subpopnames[isubpop], nn), Indicator = rep(varnames[ivar], nn),
                temp.cat)
            nrow <- nn
        }
        else {
            nn <- dim(temp.cat)[1]
            catsum <- rbind(catsum, data.frame(Type = rep(typhenames[itype], nn),
                Subpopulation = rep(subpopnames[isubpop], nn), Indicator = rep(
                varnames[ivar], nn), temp.cat, row.names = (nrow + 1):(nrow + nn)))
            nrow <- nrow + nn
        }
    }
}
}
}
# Return the data frame
catsum

```

APPENDIX F—CONTINUOUS ANALYSIS CODE

```
#####  
# Function:    cont.analysis  
# Programmers: Tony Olsen  
#              Tom Kincaid  
# Date:       March 31, 2005  
# Description:  
#   This function organizes input and output for analysis of continuous data  
#   generated by a probability survey. Input can be either an object of class  
#   psurvey.analysis (see the documentation for function psurvey.analysis)  
#   or through use of the other arguments to this function.  
#   Input:  
#     sites = a data frame consisting of two variables: the first variable is  
#             site IDs, and the second variable is a logical vector indicating which  
#             sites to use in the analysis. If psurvey.obj is not provided, then  
#             this argument is required. The default is NULL.  
#     subpop = a data frame describing sets of populations and subpopulations  
#             for which estimates will be calculated. The first variable is site  
#             IDs. Each subsequent variable identifies a Type of population, where  
#             the variable name is used to identify Type. A Type variable  
#             identifies each site with one of the subpopulations of that Type. If  
#             psurvey.obj is not provided, then this argument is required. The  
#             default is NULL.  
#     design = a data frame consisting of design variables. If psurvey.obj is  
#             not provided, then this argument is required. The default is NULL.  
#             Variables should be named as follows:  
#               siteID = site IDs  
#               wgt = final adjusted weights, which are either the weights for a  
#                   single-stage sample or the stage two weights for a two-stage  
#                   sample  
#               xcoord = x-coordinates for location, which are either the x-  
#                   coordinates for a single-stage sample or the stage two x-  
#                   coordinates for a two-stage sample  
#               ycoord = y-coordinates for location, which are either the y-  
#                   coordinates for a single-stage sample or the stage two y-  
#                   coordinates for a two-stage sample  
#               stratum = the stratum codes  
#               cluster = the stage one sampling unit (primary sampling unit or  
#                   cluster) codes  
#               wgt1 = final adjusted stage one weights  
#               xcoord1 = the stage one x-coordinates for location
```

```

#         ycoord1 = the stage one y-coordinates for location
# data.cont = a data frame of continuous response variables. The first
# variable is site IDs. Subsequent variables are response variables.
# Missing data (NA) is allowed. The default is NULL.
# sigma = measurement error variance. This variable must be a vector
# containing a value for each response variable and must have the names
# attribute set to identify the response variable names. Missing data
# (NA) is allowed. The default is NULL.
# var.sigma = variance of the measurement error variance. This variable
# must be a vector containing a value for each response variable and
# must have the names attribute set to identify the response variable
# names. Missing data (NA) is allowed. The default is NULL.
# N.cluster = the number of stage one sampling units in the resource, which
# is required for calculation of finite and continuous population
# correction factors for a two-stage sample. For a stratified sample
# this variable must be a vector containing a value for each stratum and
# must have the names attribute set to identify the stratum codes. The
# default is NULL.
# popsize = the known size of the resource - the total number of sampling
# units of a finite resource or the measure of an extensive resource,
# which is required for calculation of finite and continuous population
# correction factors for a single-stage sample. This argument is also
# used to adjust estimators for the known size of a resource. The
# argument must be in the form of a list containing an entry for each
# population Type in the subpop data frame, where NULL is a valid entry
# for a population Type. The list must be named using the variable
# names for population Types in subpop. If a population Type doesn't
# contain subpopulations, then the element of the list is either a
# single value for an unstratified sample or a vector containing a value
# for each stratum for a stratified sample, where the vector must have
# the names attribute set to identify the stratum codes. If a
# population Type contains subpopulations, then the element of the list
# is a list containing an element for each subpopulation, where the list
# is named using the subpopulation names. The element for each
# subpopulation will be either a single value for an unstratified sample
# or a named vector of values for a stratified sample. The default is
# NULL.
# stage1size = the known size of the stage one sampling units of a two-
# stage sample, which is required for calculation of finite and
# continuous population correction factors for a two-stage sample and
# must have the names attribute set to identify the stage one sampling
# unit codes. For a stratified sample, the names attribute must be set
# to identify both stratum codes and stage one sampling unit codes using
# a convention where the two codes are separated by the # symbol, e.g.,
# "Stratum 1#Cluster 1". The default is NULL.

```

```

# support = the support value for each site - the value one (1) for a
# site from a finite resource or the measure of the sampling unit
# associated with a site from an extensive resource, which is required
# for calculation of finite and continuous population correction
# factors. The default is NULL.
# swgt = the size-weight for each site, which is the stage two size-weight
# for a two-stage sample. The default is NULL.
# swgt1 = the stage one size-weight for each site. The default is NULL.
# unitsize = the known sum of the size-weights of the resource. The
# argument must be in the form of a list containing an entry for each
# population Type in the subpop data frame, where NULL is a valid entry
# for a population Type. The list must be named using the variable
# names for population Types in subpop. If a population Type doesn't
# contain subpopulations, then the element of the list is either a
# single value for an unstratified sample or a vector containing a value
# for each stratum for a stratified sample, where the vector must have
# the names attribute set to identify the stratum codes. If a
# population Type contains subpopulations, then the element of the list
# is a list containing an element for each subpopulation, where the list
# is named using the subpopulation names. The element for each
# subpopulation will be either a single value for an unstratified sample
# or a named vector of values for a stratified sample. The default is
# NULL.
# vartype = the choice of variance estimator, where "Local" = local mean
# estimator and "SRS" = SRS estimator. The default is "Local".
# conf = the confidence level. The default is 95%.
# pctval = the set of values at which percentiles are estimated. The
# default set is: {5, 25, 50, 75, 95}.
# psurvey.obj = A list of class psurvey.analysis that was produced by the
# function psurvey.analysis. Depending on input to that function,
# some elements of the list may be NULL. The default is NULL.
# Output:
# A list containing either three or five data frames of population
# estimates for all combinations of population Types, subpopulations within
# Types, and response variables. The data frames containing deconvoluted
# CDF estimates and deconvoluted percentile estimates are only included in
# the output list when an input value for measurement error variance is
# provided to the function. CDF and percentile estimates are calculated
# for both proportion and size of the population. Standard error estimates
# and confidence interval estimates also are calculated.
# The five data frames are:
# CDF - a data frame containing the CDF estimates
# Pct - a data frame containing the percentile estimates
# CDF.D - a data frame containing the deconvoluted CDF estimates
# Pct.D - a data frame containing the deconvoluted percentile estimates

```

```

#       Tot - a data frame containing the total, mean, standard deviation,
#           and variance estimates
#       If an input value for measurement error variance is not provided to the
#       function, then CDF.D and Pct.D are assigned the value NULL.
# Other Functions Required:
#   dframe.check - check site IDs, the sites data frame, the subpop data
#                   frame, the data.cat data frame, type.cat, and the data.cont data frame
#                   to assure valid contents and, as necessary, create the sites data
#                   frame, the subpop data frame, and type.cat
#   vecprint - takes an input vector and outputs a character string with
#               line breaks inserted
#   input.check - check input values for errors, consistency, and
#                 compatibility with psurvey.analysis analytical functions
#   cdf.est - estimate the cumulative distribution function (CDF) for the
#             proportion (expressed as percent) and the total of a response variable
#   cdf.decon - estimate the deconvoluted CDF for the proportion and the
#              total of a response variable
#   total.est - estimate the population total, mean, variance, and
#              standard deviation of a response variable
# Examples:
#   mysiteID <- paste("Site", 1:25, sep="")
#   mysites <- data.frame(siteID=mysiteID, Active=rep(TRUE, 25))
#   mysubpop <- data.frame(siteID=mysiteID, All.Sites=rep("All Sites",25))
#   mydesign <- data.frame(siteID=mysiteID, wgt=runif(25, 10, 100),
#                         xcoord=runif(25), ycoord=runif(25))
#   ContVar <- rnorm(25, 10, 1)
#   mydata.cont <- data.frame(siteID=mysiteID, ContVar=ContVar)
#   cont.analysis(sites=mysites, subpop=mysubpop, design=mydesign,
#                 data.cont=mydata.cont)
#
#   mydata.cont <- data.frame(siteID=mysiteID, ContVar=ContVar,
#                             ContVar.1=ContVar + rnorm(25, 0, sqrt(0.25)),
#                             ContVar.2=ContVar + rnorm(25, 0, sqrt(0.50)))
#   mysigma <- c(NA, 0.25, 0.50)
#   names(mysigma) <- c("ContVar", "ContVar.1", "ContVar.2")
#   cont.analysis(sites=mysites, subpop=mysubpop, design=mydesign,
#                 data.cont=mydata.cont, sigma=mysigma)
#####
# Begin the section when an object of class "psurvey.analysis" was input to the
# function
if(inherits(psurvey.obj, "psurvey.analysis")) {
  # Assign variables from the input list
  sites <- psurvey.obj$sites
  subpop <- psurvey.obj$subpop
  design <- psurvey.obj$design

```

```

data.cont <- psurvey.obj$data.cont
sigma <- psurvey.obj$sigma
var.sigma <- psurvey.obj$var.sigma
N.cluster <- psurvey.obj$N.cluster
popsize <- psurvey.obj$popsize
stage1size <- psurvey.obj$stage1size
support <- psurvey.obj$support
swgt <- psurvey.obj$swgt
swgt1 <- psurvey.obj$swgt1
unitsize <- psurvey.obj$unitsize
stratum.ind <- psurvey.obj$stratum.ind
cluster.ind <- psurvey.obj$cluster.ind
pcfactor.ind <- psurvey.obj$pcfactor.ind
swgt.ind.ind <- psurvey.obj$swgt.ind.ind
vartype <- psurvey.obj$vartype
conf <- psurvey.obj$conf
pctval <- psurvey.obj$pctval
}
else {
  # Check that the required data frames have been provided
  if(is.null(sites)) stop("\nThe sites data frame must be provided.")
  if(!is.data.frame(sites))
    stop("\nThe sites argument must be a data frame.")
  if(is.null(subpop))
    stop("\nThe subpop data frame must be provided.")
  if(!is.data.frame(subpop))
    stop("\nThe subpop argument must be a data frame.")
  if(is.null(design))
    stop("\nThe design data frame must be provided.")
  if(!is.data.frame(design))
    stop("\nThe design argument must be a data frame.")
  if(is.null(data.cont))
    stop("\nThe data.cont data frame must be provided.")
  if(!is.data.frame(data.cont))
    stop("\nThe data.cont argument must be a data frame.")
  # Check the design data frame for required names
  design.names <- names(design)
  temp <- match(design.names, c("siteID", "wgt", "xcoord", "ycoord", "stratum", "cluster",
    "wgt1", "xcoord1", "ycoord1"),
    nomatch = 0)
  if(any(temp == 0))
    stop("\nThe names used in the design data frame do not match the required names.")
  # Check site IDs, the sites data frame, the subpop data frame, and the data.cont
  # data frame to assure valid contents

```

```

        temp <- dframe.check(dim(design)[1], design$siteID, sites, subpop, NULL, data.cont,
design.names)
sites <- temp$sites
subpop <- temp$subpop
data.cont <- temp$data.cont
        # As necessary, remove rows of the design, sites, subpop, and data.cont data
# frames and elements of the support, swgt, and swgt1 vectors based on the
# logical variable in the sites data frame
subpop <- subpop[sites[, 2], ]
design <- design[sites[, 2], ]
data.cont <- data.cont[sites[, 2], ]
if(!is.null(support))
    support <- support[sites[, 2]]
if(!is.null(swgt))
    swgt <- swgt[sites[, 2]]
if(!is.null(swgt1))
    swgt1 <- swgt1[sites[, 2]]
sites <- sites[sites[, 2], ]
# Assign variables from the design data frame
siteID <- design$siteID
wgt <- design$wgt
xcoord <- design$xcoord
ycoord <- design$ycoord
stratum <- design$stratum
cluster <- design$cluster
wgt1 <- design$wgt1
xcoord1 <- design$xcoord1
ycoord1 <- design$ycoord1
# Determine whether the sample is stratified
stratum.ind <- length(unique(stratum)) > 1
# If the sample is stratified, convert stratum to a factor, determine stratum
# levels, and calculate number of strata
if(stratum.ind) {
    stratum <- factor(stratum)
    stratum.levels <- levels(stratum)
    nstrata <- length(stratum.levels)
}
else {
    stratum.levels <- NULL
    nstrata <- NULL
}
# Determine whether the sample has two stages
cluster.ind <- length(unique(cluster)) > 1
# If the sample has two stages, convert cluster to a factor, determine cluster
# levels, and calculate number of clusters

```

```

if(cluster.ind) {
  if(stratum.ind) {
    cluster.in <- cluster
    cluster <- tapply(cluster, stratum, factor)
    cluster.levels <- sapply(cluster, levels, simplify = FALSE)
    ncluster <- sapply(cluster.levels, length)
  }
  else {
    cluster <- factor(cluster)
    cluster.levels <- levels(cluster)
    ncluster <- length(cluster.levels)
  }
}
# Determine whether the population correction factor is to be used
pcfactor.ind <- !is.null(support)
# Determine whether the sample uses size-weights
swgt.ind <- !is.null(swgt)
# Check the vector of measurement error variance values for correct names
if(!is.null(sigma)) {
  temp.names <- names(data.cont)[-1]
  if(length(sigma) != length(temp.names))
    stop("\nThe vector of measurement error variance values is not the correct length.")
  if(is.null(names(sigma)))
    stop("\nThe vector of measurement error variance values must be named.")
  temp <- match(temp.names, names(sigma), nomatch = 0)
  if(any(temp == 0)) {
    temp.str <- vecprint(temp.names[temp == 0])
    stop(paste(
      "\nThe following names for the response variables do not occur among the
names for \nthe vector of measurement error variance values:\n",
      temp.str, sep = ""))
  }
  temp <- match(names(sigma), temp.names, nomatch = 0)
  if(any(temp == 0)) {
    temp.str <- vecprint(names(sigma)[temp == 0])
    stop(paste(
      "\nThe following names for the vector of measurement error variance
values do not \noccur among the names for the response variables:\n",
      temp.str, sep = ""))
  }
  sigma <- sigma[temp]
}
# Check the vector of values for variance of the measurement error variance for
# correct names
if(!is.null(var.sigma)) {

```

```

if(length(var.sigma) != length(temp.names))
  stop("\n\nThe vector of values for variance of the measurement error variance is
not the \ncorrect length.")
if(is.null(names(var.sigma)))
  stop("\n\nThe vector of values for variance of the measurement error variance
must be \nnamed.")
temp <- match(temp.names, names(var.sigma), nomatch = 0)
if(any(temp == 0)) {
  temp.str <- vecprint(temp.names[temp == 0])
  stop(paste(
    "\n\nThe following names for the response variables do not occur among the
names for \nthe vector of values for variance of the measurement error variance:\n",
    temp.str, sep = ""))
}
temp <- match(names(var.sigma), temp.names, nomatch = 0)
if(any(temp == 0)) {
  temp.str <- vecprint(names(var.sigma)[temp == 0])
  stop(paste(
    "\n\nThe following names for the vector of values for variance of the
measurement \nerror variance do not occur among the names for the response variables:\n",
    temp.str, sep = ""))
}
var.sigma <- var.sigma[temp]
}
# Determine the number of response values
nresp <- dim(design)[1]
# Check for compatibility of input values
temp <- input.check(nresp, wgt, sigma, var.sigma, xcoord, ycoord, stratum.ind, stratum,
  stratum.levels, nstrata, cluster.ind,
  cluster, cluster.levels, ncluster, N.cluster, wgt1, xcoord1, ycoord1, popsize, stage1size,
pcfactor.ind, support,
  swgt.ind, swgt, swgt1, unitsize, vartype, conf, pctval = pctval, subpop = subpop)
N.cluster <- temp$N.cluster
popsize <- temp$popsize
stage1size <- temp$stage1size
unitsize <- temp$unitsize
# If the sample was stratified and had two stages, then reset cluster to its
# input value
if(stratum.ind && cluster.ind) cluster <- cluster.in
# As necessary, assign missing values to the design variables
if(is.null(xcoord)) xcoord <- rep(NA, nresp)
if(is.null(ycoord))
  ycoord <- rep(NA, nresp)
if(is.null(stratum))
  stratum <- rep(NA, nresp)

```

```

if(is.null(cluster))
  cluster <- rep(NA, nresp)
if(is.null(wgt1))
  wgt1 <- rep(NA, nresp)
if(is.null(xcoord1))
  xcoord1 <- rep(NA, nresp)
if(is.null(ycoord1))
  ycoord1 <- rep(NA, nresp)
# Recreate the design data frame
design <- data.frame(siteID = siteID, wgt = wgt, xcoord = xcoord, ycoord = ycoord, stratum =
stratum, cluster = cluster, wgt1
                      = wgt1, xcoord1 = xcoord1, ycoord1 = ycoord1)
}
# Loop through all response variables
nvar <- dim(data.cont)[2]
varname <- names(data.cont)
ntypes <- dim(subpop)[2]
typenames <- names(subpop)
nrow1a <- 0
nrow2a <- 0
for(ivar in 2:nvar) {
  # Find unique data values across all subpopulations of this type of population
  cdfval <- sort(unique(data.cont[!is.na(data.cont[, ivar]), ivar]))
  # Loop through all types of populations
  for(itype in 2:ntypes) {
    # Find unique subpopulations of this type of population
    subpopnames <- levels(factor(subpop[, itype]))
    # Loop through all subpopulations of this type
    for(isubpop in 1:length(subpopnames)) {
      # Select sites in a subpopulation
      subpop.ind <- subpop[, itype] == subpopnames[isubpop]
      # Determine whether the subpopulation is empty
      if(all(is.na(data.cont[subpop.ind, ivar]))) {
        warning(paste("Subpopulation", subpopnames[isubpop], "of population
type", typenames[itype],
                      "for indicator", varname[ivar], "\ncontains no data.\n"))
        next
      }
      # Determine whether the subpopulation contains a single value
      if(sum(!is.na(data.cont[subpop.ind, ivar])) == 1) {
        warning(paste("Subpopulation", subpopnames[isubpop], "of population
type", typenames[itype],
                      "for indicator", varname[ivar], "\ncontains a single value. No
analysis was performed.\n"))
        next
      }
    }
  }
}

```

```

    }
    # For a stratified sample, remove values from N.cluster, popsize,
    stage1size,
    # and unitsize for strata that do not occur in the response variable
    if(stratum.ind) {
        temp.N.cluster <- N.cluster[!is.na(match(names(N.cluster),
unique(design[subpop.ind, 5])))]
        temp.popsiz <- popsize[[itype -
1]][[isubpop]][!is.na(match(names(popsiz[[itype - 1]][[isubpop]]),
unique(design[subpop.ind, 5])))]
        temp.stage1size <- stage1size[!is.na(match(names(stage1size),
unique(design[subpop.ind, 5])))]
        temp.unitsize <- unitsize[!is.na(match(names(unitsize),
unique(design[subpop.ind, 5])))]
    }
    else {
        temp.N.cluster <- N.cluster
        temp.popsiz <- popsize[[itype - 1]][[isubpop]]
        temp.stage1size <- stage1size
        temp.unitsize <- unitsize
    }
    # Calculate estimates for the response variable
    temp1.cont <- cdf.est(z = data.cont[subpop.ind, ivar], wgt = design[subpop.ind, 2], x
= design[subpop.ind,
        3], y = design[subpop.ind, 4], stratum = design[subpop.ind, 5], cluster =
design[subpop.ind, 6],
        N.cluster = temp.N.cluster, wgt1 = design[subpop.ind, 7], x1 =
design[subpop.ind, 8], y1 = design[
        subpop.ind, 9], popsize = temp.popsiz, stage1size = temp.stage1size, support =
support[subpop.ind],
        swgt = swgt[subpop.ind], swgt1 = swgt1[subpop.ind], unitsize = temp.unitsize,
vartype = vartype, conf
        = conf, cdfval = cdfval, pctval = pctval, check.ind = FALSE)
    if(!is.null(sigma) && !is.na(sigma[ivar - 1]))
        temp2.cont <- cdf.decon(z = data.cont[subpop.ind, ivar], wgt =
design[subpop.ind, 2], sigma = sigma[
        ivar - 1], var.sigma = var.sigma[ivar - 1], x = design[subpop.ind, 3], y
= design[subpop.ind,
        4], stratum = design[subpop.ind, 5], cluster = design[subpop.ind, 6],
N.cluster =
        temp.N.cluster, wgt1 = design[subpop.ind, 7], x1 = design[subpop.ind, 8],
y1 = design[
        subpop.ind, 9], popsize = temp.popsiz, stage1size = temp.stage1size,
support = support[

```

```

                                subpop.ind], swgt = swgt[subpop.ind], swgt1 = swgt1[subpop.ind], unitsize
= temp.unitsize,
                                vartype = vartype, conf = conf, cdfval = cdfval, pctval = pctval,
check.ind = FALSE)
                                temp3.cont <- total.est(z = data.cont[subpop.ind, ivar], wgt = design[subpop.ind, 2],
x = design[subpop.ind,
                                3], y = design[subpop.ind, 4], stratum = design[subpop.ind, 5], cluster =
design[subpop.ind, 6],
                                N.cluster = temp.N.cluster, wgt1 = design[subpop.ind, 7], x1 =
design[subpop.ind, 8], y1 = design[
                                subpop.ind, 9], popsize = temp.popsize, stage1size = temp.stage1size, support =
support[subpop.ind],
                                swgt = swgt[subpop.ind], swgt1 = swgt1[subpop.ind], unitsize = temp.unitsize,
vartype = vartype, conf
                                = conf, check.ind = FALSE)
                                # Assign estimates for the response variable to data frames
                                if(nrow1a == 0) {
                                    nn <- dim(temp1.cont$CDF)[1]
                                    cdfsum <- data.frame(Type = rep(typhenames[itype], nn), Subpopulation =
rep(subpopnames[isubpop], nn),
                                    Indicator = rep(varname[ivar], nn), temp1.cont$CDF)
                                    nrow1a <- nn
                                    nn <- dim(temp1.cont$Pct)[1]
                                    pctsum <- data.frame(Type = rep(typhenames[itype], nn), Subpopulation =
rep(subpopnames[isubpop], nn),
                                    Indicator = rep(varname[ivar], nn), temp1.cont$Pct)
                                    nrow1b <- nn
                                    nn <- dim(temp3.cont)[1]
                                    totsum <- data.frame(Type = rep(typhenames[itype], nn), Subpopulation =
rep(subpopnames[isubpop], nn),
                                    Indicator = rep(varname[ivar], nn), temp3.cont)
                                    nrow3 <- nn
                                }
                                else {
                                    nn <- dim(temp1.cont$CDF)[1]
                                    cdfsum <- rbind(cdfsum, data.frame(Type = rep(typhenames[itype], nn),
Subpopulation = rep(subpopnames[
                                    isubpop], nn), Indicator = rep(varname[ivar], nn), temp1.cont$CDF,
row.names = (nrow1a + 1):
                                    (nrow1a + nn)))
                                    nrow1a <- nrow1a + nn
                                    nn <- dim(temp1.cont$Pct)[1]
                                    pctsum <- rbind(pctsum, data.frame(Type = rep(typhenames[itype], nn),
Subpopulation = rep(subpopnames[

```

```

                                isubpop], nn), Indicator = rep(varname[ivar], nn), temp1.cont$Pct,
row.names = (nrow1b + 1):
                                (nrow1b + nn)))
                                nrow1b <- nrow1b + nn
                                nn <- dim(temp3.cont)[1]
                                totsum <- rbind(totsum, data.frame(Type = rep(typhenames[itype], nn),
Subpopulation = rep(subpopnames[
                                isubpop], nn), Indicator = rep(varname[ivar], nn), temp3.cont, row.names
= (nrow3 + 1):(nrow3 +
                                nn)))
                                nrow3 <- nrow3 + nn
                                }
                                if(!is.null(sigma) && !is.na(sigma[ivar - 1])) {
                                if(nrow2a == 0) {
                                nn <- dim(temp2.cont$CDF)[1]
                                cdfsum.D <- data.frame(Type = rep(typhenames[itype], nn), Subpopulation =
rep(subpopnames[
                                isubpop], nn), Indicator = rep(varname[ivar], nn), temp2.cont$CDF)
                                nrow2a <- nn
                                nn <- dim(temp2.cont$Pct)[1]
                                pctsum.D <- data.frame(Type = rep(typhenames[itype], nn), Subpopulation =
rep(subpopnames[
                                isubpop], nn), Indicator = rep(varname[ivar], nn), temp2.cont$Pct)
                                nrow2b <- nn
                                }
                                else {
                                nn <- dim(temp2.cont$CDF)[1]
                                cdfsum.D <- rbind(cdfsum.D, data.frame(Type = rep(typhenames[itype], nn),
Subpopulation = rep(
                                subpopnames[isubpop], nn), Indicator = rep(varname[ivar], nn),
temp2.cont$CDF,
                                row.names = (nrow2a + 1):(nrow2a + nn)))
                                nrow2a <- nrow2a + nn
                                nn <- dim(temp2.cont$Pct)[1]
                                pctsum.D <- rbind(pctsum.D, data.frame(Type = rep(typhenames[itype], nn),
Subpopulation = rep(
                                subpopnames[isubpop], nn), Indicator = rep(varname[ivar], nn),
temp2.cont$Pct,
                                row.names = (nrow2b + 1):(nrow2b + nn)))
                                nrow2b <- nrow2b + nn
                                }
                                }
                                }
                                }
}

```

```
    # Return the list
    if(!is.null(sigma)) list(CDF = cdfsum, Pct = pctsum, CDF.D = cdfsum.D, Pct.D = pctsum.D, Tot = totsum)
else list(CDF = cdfsum, Pct =
          pctsum, CDF.D = NULL, Pct.D = NULL, Tot = totsum)
}
```