Paper 040-2013

Have it Your Way: Creating Reports with the Data Step Report Writing Interface

Pete Lund, Looking Glass Analytics, Olympia, WA

ABSTRACT

SAS© provides some powerful, flexible tools for creating reports, like PROC REPORT and PROC TABULATE. With the advent of the Output Delivery System (ODS) you have almost total control over how the output from those procedures looks. But, there are still times where you need (or want) just a little more and that's where the Report Writing Interface (RWI) can help.

The Report Writing Interface is just a fancy way of saying you're using the ODSOUT object in a data step. This object allows you to layout the page, create tables, embed images, add titles and footnotes and more – all from within a data step, using whatever data step logic you need. Also, all the style capabilities of ODS are available to you so that your data step created output can have fonts, sizes, colors, backgrounds and borders to make your report look just like you want.

This presentation will quickly cover some of the basics of using the ODSOUT object and then walk through some of the techniques to create four "real world" examples. Who knows, you might even go home and replace some of your PROC REPORT code – I know I have!

THE BASICS OF RWI

PUT statements can still be used in DATA _NULL_ reporting to create reports in a DATA STEP and, with all that's available with ODS styles, they can look very nice (see Lund, 2011 for some discussion of this). But, in the new world of DATA _NULL_ reporting tables can be defined right in the data step code and even produce many different tables on the same page of output. Before a discussion of the Report Writing Interface (RWI), please understand that this paper is just to get your interest piqued. This is huge topic and you can get much more information in Dan O'Connor's 2009 SAS Global Forum paper listed in the references section. It contains 40 pages on this topic alone.

In all the example code that follows a couple assumptions are made (unless noted otherwise to expound on the example):

1. They are all within a data step, so no DATA…; or RUN; statements will be shown
2. Often code that has already been shown and discussed will not be repeated
3. All of these examples would be creating PDF files, so the "ODS sandwich" statements will be shown (ODS PDF file=…; and ODS PDF CLOSE;)

The RWI uses a data step object called ODSOut. There are "methods" (like functions) of that object that will create tables, rows, cells, text, page breaks, lines, etc. To use an ODSOut object it is first declared and given a name – this only has to be done once in the data step and is routinely placed in a conditional section of code:

```
if _n_ eq 1 then
  do;
    declare odsout t();
    <other statements>
  end;
```

This gives us an ODSOut object named "t" – we'll use that name to reference methods that build our output.

Again, the DECLARE statement only have to be executed once in the data step.

Once the object is declared you can call "methods" that perform different tasks. For instance, with our object "t," just a few of the possible methods are:

t.table_start()	- begins a table (there is a table_end method that closes a table)
t.row_start()	- begins a row in that table – you can have as many rows in the table as you want (there is also a row_end method that closes a row)
t.format_cell()	- inserts a cell (column) into that row – you can have as many cells in a row as you want, but each row must have the same number of cells
t.format_text()	- inserts a line of text (not part of a table)
t.line()	- puts a horizontal line on the page
t.page()	- inserts a page break
t.title()	- creates a page title (there is also a footnote method that creates a page footnote)

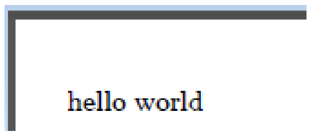
Note that all of these methods calls have parentheses, which are required – even if empty. There are parameters that can be placed in the parentheses. For example, the Format_Cell method has a "text" parameter that specifies the text to be printed in the cell. You can specify style attributes in most method calls as well, specifying cell borders, appearance of text, line widths, etc., with the OVERRIDES or STYLE parameters. These will be discussed in more detail as we move along.

Here's a very simple example of using the ODSOUT object. No dataset will be read by the data step, so wrapping the DECLARE statement in an _N_ = 1 loop is not necessary, because the step will execute only once.

```
data _null_;
  declare odsout t();

  t.format_text(text: 'Hello World);
run;
```

This method simply writes text to the page. By default is will be left justified and use the font attributes associated with the body of the ODS style in effect.

The output from this step would be a single page with the words in the upper left corner. Note that if there are any TITLE or FOOTNOTE statements that are still active, these would also appear on the page and the text would be after the titles.



CREATING SIMPLE TABLES

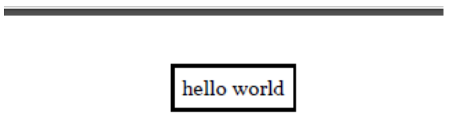
But, reports are usually not comprised of just text. The RWI can define tables of data and there are sets of table, row and cell methods that allow us to do that.

```
t.table_start();
  t.row_start();
  t.format_cell(text: 'Hello World);
  t.row_end();
t.table_end();
```

The creation of a table is pretty straightforward:

- *Start the table*
- *Start the first, and only, row*
- *Insert a single cell with some text*
- *End the row*
- *End the table*

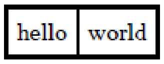
As noted above, the parentheses following the method calls are required, even if no parameters are passed. Also note that the FORMAT_CELL method parameter is exactly the same as that in the FORMAT_TEXT method call in the earlier example. But, the appearance of the output will be very different. The code above, rather than just creating text on the page, would create a single cell "table" with the words "Hello World". By default, the output will be centered on the page, again following any titles. The cell borders, fonts, color, etc. will be determined by the ODS STYLE that is being used on the ODS PDF statement.



It's simple to make the tables more table-like, by adding more columns and rows. Getting more columns is just a simple matter of having more FORMAT_CELL calls between the ROW_START and ROW_END. Note that each cell is bordered individually, just as would be expected in a table.

```
t.row_start();
  t.format_cell(text: 'hello');
  t.format_cell(text: 'world');
t.row_end();
```

A row can have any number of cells – here we're creating a row with two cells (columns), each with a single word

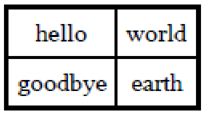


hello	world

Getting more rows is just as simple – add as many ROW_START…ROW_END blocks needed between the TABLE_START and TABLE_END.

```
t.row_start();
  t.format_cell(text: 'hello');
  t.format_cell(text: 'world');
t.row_end();
t.row_start();
  t.format_cell(text: 'goodbye');
  t.format_cell(text: 'earth');
t.row_end();
```

A table can have any number of rows – here we've added a second row to the previous example

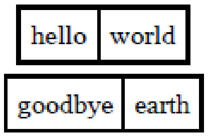


hello	world
goodbye	earth

Notice above that the columns are automatically sized to fit the largest text in the column in the entire table. To illustrate this further, and to show that in addition to multiple columns and rows, multiple tables can also be created in the same data step.

```
t.table_start();
  t.row_start();
    t.format_cell(text: 'hello');
    t.format_cell(text: 'world');
  t.row_end();
t.table_end();
t.table_start();
  t.row_start();
    t.format_cell(text: 'goodbye');
    t.format_cell(text: 'earth');
  t.row_end();
t.table_end();
```

These are the same two rows we had before, but now they are in separate tables. Notice that the sizing of the columns is table-specific and the columns in the two rows are no longer the same width and that there is now separation between the rows.



hello	world

goodbye	earth

The two rows are no longer "joined" and widths of the columns are different. Later, ways to control lots of attributes of the tables, which could have made these two tables look much the same, will be shown.

DATA-DRIVEN TABLES

It's not too practical to think of hard-coding all the data to be presented in a table. Fortunately, in addition to a quoted string, the TEXT attribute of the FORMAT_CELL method shown in the examples above can take a variable or expression as its value. This allows for creation of tables from data in datasets or from variables created in the data step. The following examples use the class list from the SASHELP.CLASS dataset.

```
set sashelp.class;

if _n_ eq 1 then declare odsout t();
```

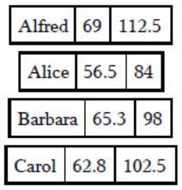
Now, bring in a dataset and we'll use values from that to populate our table

Remember, only need to declare the ODSOUT object once – do it on the first iteration of the data step.

```
  t.table_start();
    t.row_start();
      t.format_cell(text: name);
      t.format_cell(text: height);
      t.format_cell(text: weight);
    t.row_end();
  t.table_end();
```

Now, rather than quoted values in the TEXT parameter of the FORMAT_CELL method call, put the name of a variable. The contents of that variable will be placed in the cell.

But, there's a slight problem with the above code – the TABLE_START and TABLE_END methods are going to be called for every iteration of the data step and the result is a separate table for every observation, as shown in the table snippet to the right. That might be what is wanted, but probably not.



Alfred	69	112.5

Alice	56.5	84

Barbara	65.3	98

Carol	62.8	102.5

The solution is simple – place the TABLE_START call in the _N_ eq 1 logic and the TABLE_END call with a condition triggered by an END= option on the SET statement.

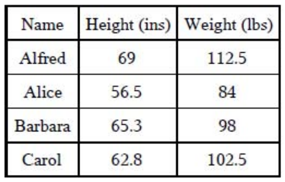
```
  set sashelp.class end=done;

  if _n_ eq 1 then
    do;
      declare odsout t();

      t.table_start();
      t.row_start();
        t.format_cell(text: 'Name');
        t.format_cell(text: 'Height (ins)');
        t.format_cell(text: 'Weight (lbs)');
      t.row_end();
    end;

    t.row_start();
      t.format_cell(text: name);
      t.format_cell(text: height);
      t.format_cell(text: weight);
    t.row_end();

  if done then t.table_end();
```

Use the END= option to define a variable that will be set to 1 (true) when the end of the dataset is reached.

In addition to declaring the ODSOUT object, move the TABLE_START call to the _N_ eq 1 block of code. Also, this is a good place to add a single header row to the table. This ROW_START and ROW_END block will only be executed once.

The row and cell code is exactly the same as before – all that needed to be changed was when the table started and stopped. Now each of these rows will be in the same table

When the end of the dataset is reached, end the table.



Name	Height (ins)	Weight (lbs)
Alfred	69	112.5
Alice	56.5	84
Barbara	65.3	98
Carol	62.8	102.5

Those couple simple changes create a single table, with all the columns the same size and the rows joined. There is a header row to tell the reader what's in the table. Now is the time to take a look at to control now just what appears in the table, but how the table appears.

Controlling the Appearance of the Tables

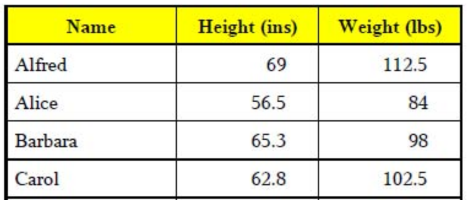
The programmer has control over almost all aspects of the appearance of the table – text attributes like font, color, size and style; cell attributes like borders, size, alignment and background; table attributes like spacing and borders. All can be controlled at most down to the tiniest detail. There are two parameters that can be used in most method calls to do this: OVERRIDES and STYLE.
The real difference between the two parameters is where the list of attributes to apply to the object is maintained. The OVERRIDES parameter lists the "overrides" of the default attributes in the method call itself, just like the TEXT parameter. The STYLE parameter references a style element that is defined in PROC TEMPLATE for the ODS STYLE that is currently in use.

A quick example will show how easy, yet powerful, this is. First, change the appearance of the header rows to set them off by overriding a few of the attributes of the cells.

The OVERRIDE parameter is used to change default attributes. These are the cells in the header row, which will be made bold, with a yellow background.

```
t.row_start();
  t.format_cell(text: 'Name',
                overrides: 'background=yellow fontweight=bold cellwidth=30mm');
  t.format_cell(text: 'Height (ins)',
                overrides: 'background=yellow fontweight=bold cellwidth=25mm');
  t.format_cell(text: 'Weight (lbs)',
                overrides: 'background=yellow fontweight=bold cellwidth=25mm');
t.row_end();
```

We can use the CELLWIDTH attribute to set the width of the columns.



Name	Height (ins)	Weight (lbs)
Alfred	69	112.5
Alice	56.5	84
Barbara	65.3	98
Carol	62.8	102.5

Each cell can have its own list of overrides – here two of them are the same across all three cells and one (cellwidth) is not. In the code below, some overrides will also be added to the data rows to left-align the name. Also, the height and weight values are right-aligned and moved a little over to the left, with the RIGHTMARGIN attribute, so that the values are not right against the edge of the cell.

```
t.row_start();
  t.format_cell(text: name,
                overrides: 'just=left');
  t.format_cell(text: height,
                overrides: 'just=right rightmargin=4mm');
  t.format_cell(text: weight,
                overrides: 'just=right rightmargin=4mm');
t.row_end();
```

In the first example of this table, all the values are centered. Here, the name is now left-justified and the other values are right-justified.

As was noted for the TEXT attribute earlier, the OVERRIDE values can be either a quoted string, as above, or a character variable (or expression). In the code above, the height and weight cells could be coded as follows, with identical results:

```
HW_over = 'just=right rightmargin=4mm';
t.format_cell(text: height, overrides: HW_over);
t.format_cell(text: weight, overrides: HW_over);
```

In this simple example, a hard-coded variable is used to set come common attributes. But, using variables instead of hard-coded attribute values also allows a dataset to contain not only the data, but information about how the data should be displayed. In one of the "real world" reports to come, at an example of this will be shown.

Another, and probably preferable way, to deal with groups of common attributes is to create an ODS style element that contains those attributes. PROC TEMPLATE is used to create the style, which will then be used in the ODS PDF statement that defines the output file.

```
proc template;
  define style test;
  parent=styles.printer;
```

Create a new ODS style, called TEST, that uses the PRINTER style as its base (that's the default style for PDF).

```
    style DataCells from body /
      just=right
      rightmargin=4mm;
    end;
  run;
```

The attributes that were in the OVERRIDES are now put in a STYLE element called DATACELLS. The BODY element in the PRINTER style is what usually defines the attributes of the text in the table. This overrides two of those values.

The following FORMAT_CELL calls will again produce the same table as those above with the OVERRIDES parameters.

```
  ods pdf file=<file reference> style=test;
```

Use the newly defined style in the ODS PDF statement.

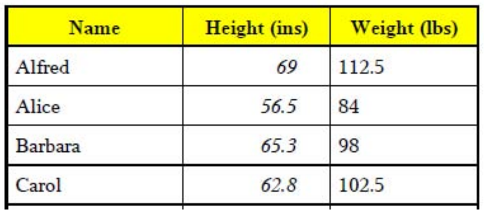
```
  <... previous data step code ...>

  t.format_cell(text: height, style: 'DataCells');
  t.format_cell(text: weight, style: 'DataCells');
```

Instead of the OVERRIDES parameter, use the STYLE parameter. The style element name is in quotes, but could also be a variable that contains the style element name.

As might expect be expected, the OVERRIDES and STYLE parameters can also be used together. If both are used, the attribute list is additive, but common attributes use the values in the OVERRIDES. In the code below, the height and weight cells have both parameters.

```
  t.format_cell(text: height, style: 'DataCells', overrides: 'fontstyle=italic');
  t.format_cell(text: weight, style: 'DataCells', overrides: 'just=left');
```

In the example above, attributes set in both places are used. The height data is still right justified, with a 4mm margin on the right. But, another attribute has been added – italic text. The weight column has one of the attributes set in the STYLE changed in the OVERRIDES parameter – the text in the cell is now left-justified instead of right justified.



Name	Height (ins)	Weight (lbs)
Alfred	*69*	112.5
Alice	*56.5*	84
Barbara	*65.3*	98
Carol	*62.8*	102.5

Being able to set attributes in both places gives a lot of control over how the output will look. Also, there's no real "right" or "wrong" way to do it. Sometimes, it is very handy to be able to see all the attribute values in the data step code, without having to look at the PROC TEMPLATE code. It is often advantageous to see the values where they're being used. If this is so, only use the STYLE parameter when there are a lot of attributes being set or there are a lot of cells with a common set of attributes.

SPANNING COLUMNS AND ROWS

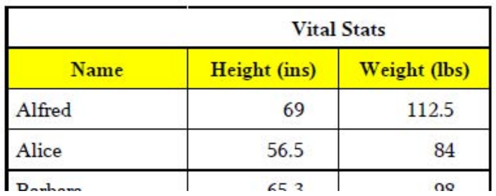
There are header cells in the table above, but information in a table can often be more understandable with cells that span multiple columns or rows that contain related information. There are two parameters in the FORMAT_CELL method that control the spanning: COLSPAN, for specifying the number of columns the cell should span, and ROWSPAN, for specifying the number of rows that a cell should span.

A single row added to the table defined above can add set the height and weight columns off a little bit from the name column. This code would immediately precede the row with the height and weight header text.

A blank cell to go over the Name column and then "Vital Stats" will be in a cell that spans the Height and Weight columns.

```
  t.row_start();
    t.format_cell();
    t.format_cell(text: 'Vital Stats',
                  overrides: 'borderleftcolor=white fontweight=bold', colspan: 2);
  t.row_end();
```

There have to be the same number of columns in each row of the table. There are three columns in the body of the table and three in the row above, taking into account the vital stats column counts as two. By default, the cell borders would have been drawn around the two merged cells, with no line in the middle.



	Vital Stats	
Name	**Height (ins)**	**Weight (lbs)**
Alfred	69	112.5
Alice	56.5	84
Barbara	65.3	98

Nothing else new here, except a new attribute: setting the BORDERLEFTCOLOR to white makes the appearance a little cleaner. As expected, the "LEFT" in this attribute could also be "RIGHT," "TOP," or "BOTTOM."

Rows can also be spanned with the ROWSPAN parameter. Suppose that the table above was sorted by gender – a far-right column could be added with the gender value and spanned so as not to repeat the gender on every row. A few things must be done to make this work correctly.

```
  by sex;
```

The table is sorted by the variable Sex, so that the rows can be grouped

In the _N_ eq 1 block, add a little to the row above the column headers.

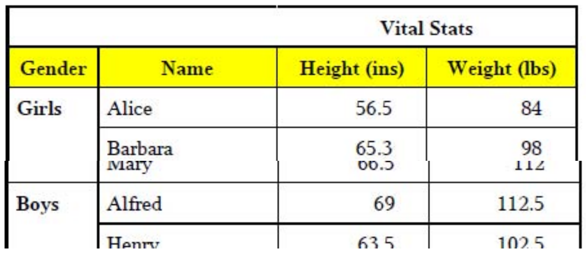
```
  t.table_start();
    t.row_start();
      t.format_cell(colspan: 2);
      t.format_cell(text: 'Vital Stats',
                    overrides: 'borderleftcolor=white fontweight=bold', colspan: 2);
    t.row_end();
```

The last example had a blank column above the Name. Now we need another blank column above Sex – we could either add another FORMAT_CELL or span this over 2 columns

In the data rows, add the gender column. Note that for the column spanning we know how many columns to span and hardcoded the values (2). But, the number of each type of row is determined by the data. In a prior step, the number of each sex value has been computed and added to each row in a variable called "num." The cell only needs to be created for the first value of each gender, when the value of FIRST.GENDER is true. This demonstrates another principle – that the method calls themselves can be conditional.

The cell will only be created on the first of a gender value.The value of Num determines how many rows will be spanned and the VJUST=TOP attribute will move the value to the top of the spanned cell.

```
  t.row_start();
    if first.sex then t.format_cell(text: put(Sex,$Sex.),
                                    overrides:'just=left fontweight=bold vjust=top',
                                    rowspan: num);
    t.format_cell(text: name,overrides: 'just=left');
    t.format_cell(text: height, style: 'DataCells');
    t.format_cell(text: weight, style: 'DataCells');
  t.row_end();
```



		Vital Stats	
Gender	**Name**	**Height (ins)**	**Weight (lbs)**
Girls	Alice	56.5	84
	Barbara	65.3	98
	Mary	66.5	112
Boys	Alfred	69	112.5
	Henry	63.5	102.5

The rest of the cells are defined just as they were before, with a new spanning row created when the value of sex changes.

The complete code for this last example, which covers everything discussed so far, appears in Appendix A, along with the final output.

Laying Out the Page

Those familiar with ODS LAYOUT and ODS REGION statements know that they can be used to place output from one or more SAS procedures anywhere on a page. A powerful feature of the RWI are analogous methods for the ODSOUT object. This means that the tables and text produced in the data step can be placed anywhere on the page.

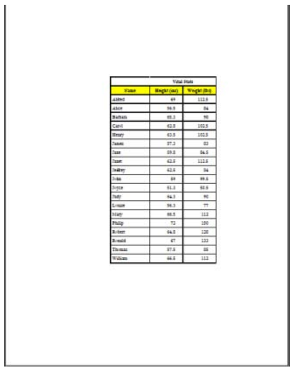
The LAYOUT_ABSOLUTE, REGION and LAYOUT_END method calls allow for tables and text created with other method calls to be placed in exact locations on the page. Like the ODS REGION statement, the REGION method call allows for X, Y, HEIGHT and WIDTH parameters defining the position and size of the region. With just a few additional lines of code, the table created above can be placed at a designated position on the page.

```
if _n_ eq 1 then
  do;
    declare odsout t();

    t.layout_absolute();

    t.region(x: '2in', y: '2in', height: '8in', width: '5in');

    <same table start and header row code as before>

  end;

    <same table row code as before>

if done then
  do;
    t.table_end();
    t.layout_end();
  end;
```

The LAYOUT_ABSOLUTE method "turns on" the ability set up regions

The upper-left corner of the table created here will start 2in from the left and 2in from the top – the height and width parameters are optional, but must be big enough to hold the generated output. If the region is too small, the output is suppressed.

After the table is ended, end the layout



	Vital Stats	
Name	Height (ins)	Weight (lbs)
Alfred	69	112.5
Alice	56.5	84
Barbara	65.3	98
Carol	62.8	102.5
Henry	63.5	102.5
James	57.3	83
Jane	59.8	84.5
Janet	62.5	112.5
Jeffrey	62.5	84
John	59	99.5
Joyce	51.3	50.5
Judy	64.3	90
Louise	56.3	77
Mary	66.5	112
Philip	72	150
Robert	64.8	128
Ronald	67	133
Thomas	57.5	85
William	66.5	112

As can be seen in the example to the right, the table has been shifted down and to the right. Note that the Y parameter is sensitive to whatever is already on the page. If there are titles or other, no "regionalized" output already on the page, this table would begin 2in below that. (Note: this is the same behavior as the ODS REGION statement.)

There is nothing to prevent multiple regions on the same page. If the class dataset were sorted by region, it would be a simple matter to have separate tables for boys and girls, side by side on the page. The complete code for this and a snapshot of the output is shown in Appendix B.

There is also a LAYOUT_GRIDDED method that, along with REGION and LAYOUT_END methods, allow for a grid to be defined on the page and output to be placed in one of the grid "cells." See Dan O'Connor's paper for examples of this technique.

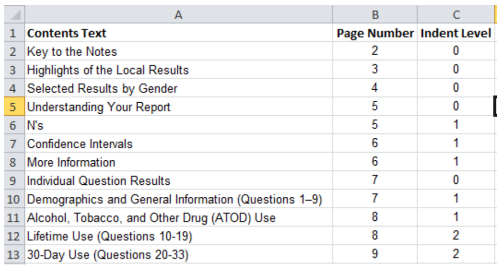
Real World Examples: A Table of Contents

Let's look at some real-life examples that use the RWI to produce the results. First, a few assumptions that will be made for all of the examples:

- All will be part of a PDF document, so the ODS PDF statement (and corresponding CLOSE) are not shown
- When needed, all use ~ as the ODS ESCAPECHAR

- For the sake of brevity, often some formatting code will be left out if it does not directly affect the current
- For the sake of clarity, hard-coded values are often shown that, in the production jobs, are actually macro calls or macro variable references

In this first example, let's build a table of contents for a report using a data step with just one RWI statement. In this case the information about the pagination of the report is stored in an Excel spreadsheet. This could have been in a SAS dataset, database table or any other source that SAS could read. (We kept it in a spreadsheet, so someone else could maintain the information.) The data step reads the spreadsheet and uses the "Context Text" and "Page Number" columns as the text in the table.



	A	B	C
1	Contents Text	Page Number	Indent Level
2	Key to the Notes	2	0
3	Highlights of the Local Results	3	0
4	Selected Results by Gender	4	0
5	Understanding Your Report	5	0
6	N's	5	1
7	Confidence Intervals	6	1
8	More Information	6	1
9	Individual Question Results	7	0
10	Demographics and General Information (Questions 1–9)	7	1
11	Alcohol, Tobacco, and Other Drug (ATOD) Use	8	1
12	Lifetime Use (Questions 10-19)	8	2
13	30-Day Use (Questions 20-33)	9	2

The simple data step below is all that's needed to create a simple table of contents. The values are used in a FORMAT_TEXT method call and written to the file.

Use the spreadsheet as the input "dataset"

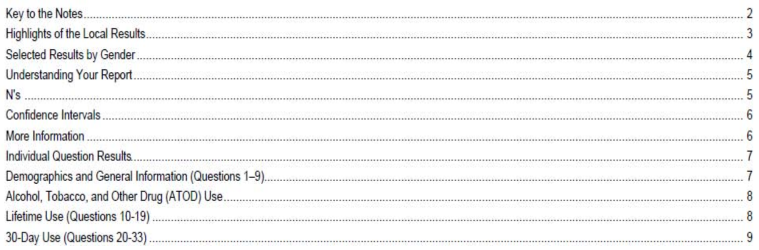
Declare the ODSOUT object

```
data _null_;
  set TOC.'Report Contents$'n;

  if _n_ eq 1 then declare odsout rt();

  LineText = catt(Contents_Text,'~{leaders .}',Page_Number);

  rt.format_text(text: LineText,overrides: "cellwidth=100pct");
run;
```

The ODS escape function LEADERS will put a line of dots between the Contents _Text and Page_Number fields

Need to tell the string to use 100% of the page width – otherwise there would only be one dot between the text and page number

To the right we see the results of the above data step. The text and page numbers are "stretched" across the whole page, with the dot leaders filling in the gap.



Key to the Notes	2
Highlights of the Local Results	3
Selected Results by Gender	4
Understanding Your Report	5
N's	5
Confidence Intervals	6
More Information	6
Individual Question Results	7
Demographics and General Information (Questions 1–9)	7
Alcohol, Tobacco, and Other Drug (ATOD) Use	8
Lifetime Use (Questions 10-19)	8
30-Day Use (Questions 20-33)	9

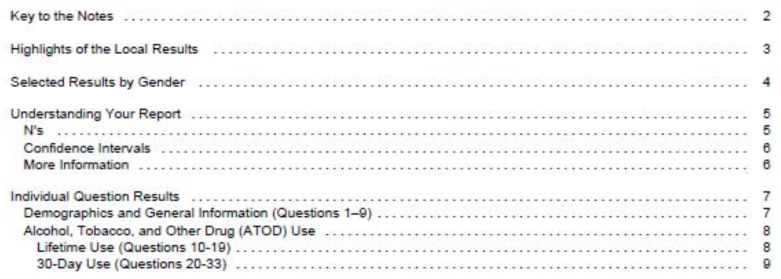
But, as menioned earlier, there are times when it's advantageous to use information in the dataset for formatting the results. Here, we can use the Indent_Level column in the spreadsheet to help make the table of contents more readable by adding just a few lines of code.

```
  LineText = catt(Contents_Text,'~{leaders .}',Page_Number);

  if Indent_Level eq 0 then LineText = catt('~{newline}',LineText);

  if Indent_Level eq 0 then LOvr = "cellwidth=100pct";
  else LOvr = catx(' ', "cellwidth=100pct",catt('marginleft=',Indent_Level*3,'mm'));

  rt.format_text(text: LineText, overrides: LOvr);
```

When Indent_Level is 0, add a line break

Move the CELLWIDTH to a variable and if the INDENT_LEVEL gt 0, add 3mm to the left margin for each indent level

The new TOC is much easier to read, with line breaks between each section and subsections indented under the main headings. All of this by using information in the data, rather than any “fancy” coding.



The complete table of contents is shown in Appendix C.

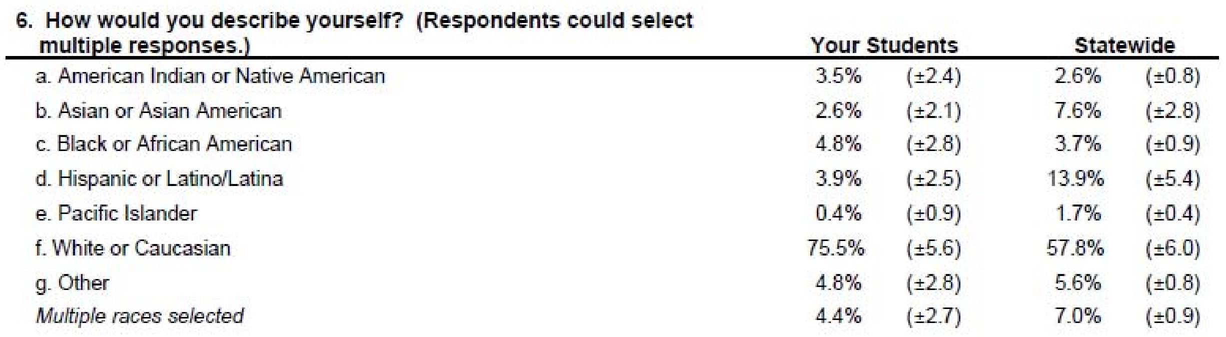
REAL WORLD EXAMPLES: “REPLACING” PROC REPORT

As seen in some of the examples above, RWI can be used to create tables that look a lot like PROCs REPORT, PRINT or TABULATE. Often times, the level of control over the appearance of the tables make RWI a great choice. Also, as shown above, the ability to use data that is not directly displayed in the table simplifies the coding that might be required in PROC REPORT, where conditional logic can also be use.

In this example, an RWI data step will produce a table that looks very much like something PROC REPORT or TABULATE could produce. The data comes from reports of a statewide survey of students, where local results (from a school, district or county) are compared to results of a statewide sample.

In the _N_ eq 1 block, the ODS object “ft” is declared, a table started and the following header row is defined. Notice that a total of eight columns are defined – three with a COLSPAN of 2 and two single-cell columns. The single-cell columns have no text and are very narrow (4mm and 7mm). They are simply used to add a little white space between the columns that contain data.

```
  ft.row_start();
    ft.format_cell(text: '6. How would you describe yourself? (Respondents could....)',
                   style: 'HeaderRows',
                   overrides: 'cellwidth=104mm just=left', colspan: 2);
    ft.format_cell(text: " ",overrides: 'cellwidth=4mm');
    ft.format_cell(text: "Your Students",style: 'HeaderRows',
                   overrides: 'cellwidth=26mm', colspan: 2);
    ft.format_cell(text: " ",overrides: 'cellwidth=7mm');
    ft.format_cell(text: "Statewide",style: 'HeaderRows',
                   overrides: 'cellwidth=26mm', colspan: 2);
  ft.row_end();
```
The header (and data rows) are shown in the table below:



6. How would you describe yourself? (Respondents could select multiple responses.)	Your Students		Statewide	
a. American Indian or Native American	3.5%	(±2.4)	2.6%	(±0.8)
b. Asian or Asian American	2.6%	(±2.1)	7.6%	(±2.8)
c. Black or African American	4.8%	(±2.8)	3.7%	(±0.9)
d. Hispanic or Latino/Latina	3.9%	(±2.5)	13.9%	(±5.4)
e. Pacific Islander	0.4%	(±0.9)	1.7%	(±0.4)
f. White or Caucasian	75.5%	(±5.6)	57.8%	(±6.0)
g. Other	4.8%	(±2.8)	5.6%	(±0.8)
Multiple races selected	4.4%	(±2.7)	7.0%	(±0.9)

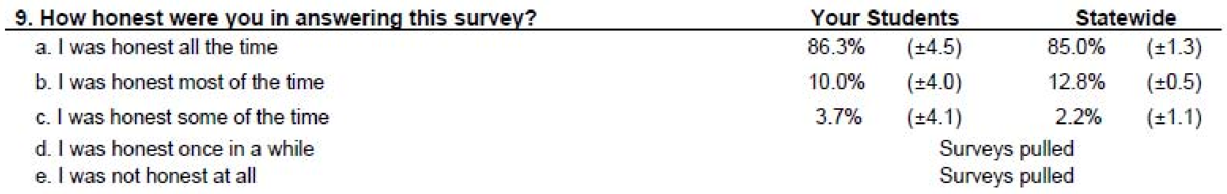
The code for the data rows is also similar to code that has been demonstrated earlier. In this case, there are actually eight columns defined, to match those defined in the header row.

```
  ft.row_start();
    ft.format_cell(text: " ",overrides: 'cellwidth=4mm');
    ft.format_cell(text: put(VarValue,Race.),overrides: "cellwidth=100mm just=left");
    ft.format_cell(text: " ");
    ft.format_cell(text: put(Percent,NoMissings.),overrides: "cellwidth=12mm");
    ft.format_cell(text: put(PlusMinus,PlusMinus.),overrides: "cellwidth=14mm");
    ft.format_cell(text: " ");
    ft.format_cell(text: put(StatePercent,NoMissings.),overrides: "cellwidth=12mm");
    ft.format_cell(text: put(StatePlusMinus,PlusMinus.),overrides: "cellwidth=14mm");
  ft.row_end();
```

Just a couple things to note here:

1. Notice that the cell widths of columns 1 and 2 (100mm and 4mm) sum to the total in the header column (104mm). The same is true of the columns 4 and 5, and 7 and 8 (12mm and 14mm) equaling the 26mm in the header.
2. The first columns, of width 4mm, is simply used to indent the values. A LEFTMARGIN attribute of 4mm in the overrides of the VarValue column, with a total CELLWIDTH of 104mm, would have produced the same results.

Now, these results would have been very easy to produce in PROC REPORT. But, there were some circumstances where the desired result would have been much more difficult. One example was a question on honesty in answering the survey. Surveys from students who responded that they were not very honest were removed from the final results, but the complete set of answers to the question were required in the report, as shown below.



9. How honest were you in answering this survey?	Your Students		Statewide	
a. I was honest all the time	86.3%	(±4.5)	85.0%	(±1.3)
b. I was honest most of the time	10.0%	(±4.0)	12.8%	(±0.5)
c. I was honest some of the time	3.7%	(±4.1)	2.2%	(±1.1)
d. I was honest once in a while	Surveys pulled			
e. I was not honest at all	Surveys pulled			

Having the "Surveys pulled" text spanning the data columns would have been a challenge in SAS procedure output. However, in the data step it is a rather trivial matter. First, the data row defined above is put in an IF…THEN conditional, based on whether a suppression flag is off. If not, we create a "data row" that spans five columns and contains the text that we want. Notice that five columns are spanned to account for the four data columns and the blank, white-space column between them.

```
  if not SuppressValues then
    do;
      <data row code from above>
    end;
  else ft.format_cell(text: 'Surveys pulled',overrides: "just=center", colspan: 5);
```

The real reports that this example comes from are over 55 pages long and contain over 240 tables generated using this method, along with other assorted tables and charts. A couple complete pages from a sample report are shown in Appendix D. Also see the AskHYS information in the reference section for links to publicly available versions of these reports.

REAL WORLD EXAMPLES: FREE-FORM REPORTS

In earlier examples, LAYOUT_ABSOLUTE was used to position RWI-generated tables at a particular location on the page. Another use of the layout capabilities of RWI is to “fill out” a form. This example comes from a web-based STD surveillance application. Data can be entered on line directly from patient records or a form can be printed, filled out by hand and the data entered at a later time, if a computer is not readily available. An example of one page of the form is shown in Appendix E-1.

Once the data is in the database, the user can also print the same form, filled with the data that was entered into the application. The RWI is used to put the data onto the page. There is some tedious initial set up, as the X-Y coordinates of all the form fields must be calculated and stored. But, once it’s done, the form can be printed for any case. As with the table of contents example, the information about the form fields is stored in a spreadsheet. Again, this is just for the ease of maintaining the information.

There are two “tricks” to make this work. The first “trick” is to get the form onto the page. It would be very difficult (never say impossible with SAS) to recreate the form, with all the rounded corners and check boxes. Besides, someone went to a lot of work to create the form, so why not use it as the basis for our report. A simple addition to the style template used to define the page is made in PROC TEMPLATE. This is a two-page report and there is a style template for each page – the templates are exactly the same, except for the background image. The images are simply image files of the form pages.

```
proc template;
  define style work.myjournal_p1;
  parent=styles.journal;
  style body / backgroundimage="<image-location>\STD_Form_1.jpg"
             margintop=0mm marginleft=0mm height=10.5in width=8in;
  end;

  define style work.myjournal_p2;
  parent=styles.journal;
  style body / backgroundimage="<image-location>\STD_Form_2.jpg"
             margintop=0mm marginleft=0mm height=10.5in width=8in;
  end;
run;
```

The template for page 1 parents off the Journal style and modifies the BODY element by adding a BACKGROUNDIMAGE attribute

The template for page 2 is the same as above, but with the image of the 2nd page of the form

On the ODS PDF statement, the page 1 style template is used and the image of the form will be the background on the page and any output created will be on top of that background image.

```
ods pdf file='<file-location>' style=work.myjournal_p1 notoc;

<code here to generate first page of output>
```

Before the code for the 2nd page starts, another ODS PDF statement is used to change the template. This will continue to write to the same file defined in the initial ODS PDF statement, but now with the other template, a different background image will be used. This method can be used for as many form pages as needed – just change the style templates between each page.

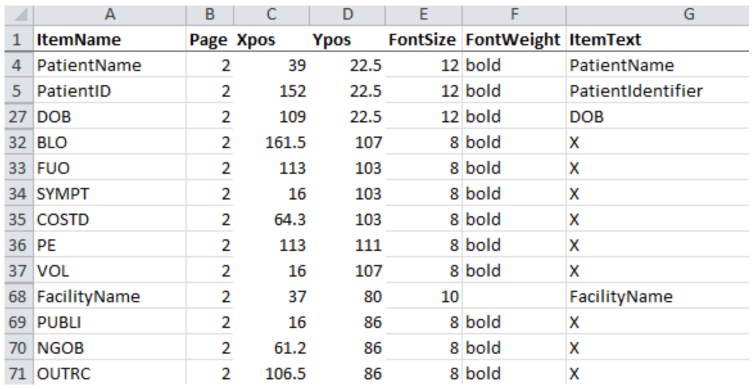
```
ods pdf style=work.myjournal_p2;
```

No FILE= here, just changing the template

The second “trick” to make this all work is to get the information about coordinates, fonts and styles into the data step. A HASH object is used to hold all the data in the spreadsheet and it is loaded in the _n_ eq 1 block, along with the declaration and initial setup of the ODSOUT object.

The information from the spreadsheet is loaded into a HASH object called "pl" - the key field is ItemName and if a search on that key is found, all the other data columns are populated.

```
if _n_ eq 1 then
  do;
    declare hash pl(dataset: "pl.'PageLayout$'n");
    pl.defineKey("ItemName");
    pl.defineData('Xpos','Ypos','ItemText','FontSize','FontWeight');
    pl.defineDone();

    <more code>
  end;
```



	A	B	C	D	E	F	G
1	ItemName	Page	Xpos	Ypos	FontSize	FontWeight	ItemText
4	PatientName	2	39	22.5	12	bold	PatientName
5	PatientID	2	152	22.5	12	bold	PatientIdentifier
27	DOB	2	109	22.5	12	bold	DOB
32	BLO	2	161.5	107	8	bold	X
33	FUO	2	113	103	8	bold	X
34	SYMPT	2	16	103	8	bold	X
35	COSTD	2	64.3	103	8	bold	X
36	PE	2	113	111	8	bold	X
37	VOL	2	16	107	8	bold	X
68	FacilityName	2	37	80	10		FacilityName
69	PUBLI	2	16	86	8	bold	X
70	NGOB	2	61.2	86	8	bold	X
71	OUTRC	2	106.5	86	8	bold	X

The Xpos and Ypos columns hold the position on the page where the output will be placed. The other columns contain the font size and weight, and the value that will be placed at that position – either an "X" in a box of the value of a variable.

This page of the form displays information about the presumptive diagnosis that caused the patient to seek treatment.

There are a number of types of information displayed on the form. A number of fields contain just a single value for the case (e.g., patient name, facility name, date of birth). These are all stored in a dataset called PatientInfo. Other variables can have multiple values for the case, such as the reason for the visit (stored in VisitReason). The database containing information on all cases has already been queried and multiple datasets have been created containing the information about the case for which the report is being generated. Multiple SET statements will be used to read all this information, but the data step that creates the page will iterate just one time, calling the %DisplayInfo macro the put the information on the page.

```
  set visit.PatientInfo end=endPatient;

  %DisplayInfo(PatientName);
  %DisplayInfo(PatientID);
  %DisplayInfo(DOB);
  %DisplayInfo(DOV);
  %DisplayInfo(FacilityName);
  <more fields displayed>

  do while (not endFacType);
    set visit.VisitReason end=endReason;
    %DisplayInfo(VisitReasonCode,GetValue=Y);
  end;

  do while (not endPD);
    set visit.VisitPD end=endPD;
    %DisplayInfo(SyndromeCode,GetValue=Y);
  end;
```

The single value information is all stored in the dataset PatientInfo and the %DisplayInfo macro is called for each variable – the variable in the macro call is looked up in the spreadsheet and the value will be printed at the location listed (e.g., the patient's name)

For those variables that can have multiple values, the SET statement is placed inside a DO WHILE loop which is [illegible] *ıld be* [illegible] *ed up in* [illegible] *ted.*

Another dataset is looped through and the values displayed on the page.

Note that there is no need for the datasets to be in any particular order as the X/Y coordinates can place the values at any location on the page.

So, how does that macro work? The ultimate purpose of the macro is to generate two method calls: REGION, to place the text where it's needed, and FORMAT_TEXT to put the proper value. The full code is listed here.

```
  %macro DisplayInfo(IN,GetValue=N);
    %if &GetValue eq N %then %str(ItemName = "&IN";);
    %else %str(ItemName = vvalue(&IN););

    rc = pl.find();
```

ItemName is the "key" of the hash – it will either be the actual value passed to the macro ("&IN") or the value of a variable name passed to the macro (vvalue(&IN))

The FIND method on the hash looks for the value in the key variable (ItemName)

If the value is found, the return code is 0, and all the data fields are filled with the associated values

```
    if not rc then
      do;
        vi.region(x: catt(Xpos,'mm'), y: catt(Ypos,'mm'));
        if missing(FontWeight) then FontWeight = 'medium';
        TextOverrides = catx(' ',catt('fontsize=',FontSize,'pt'),catt('fontweight=',FontWeight));
        vi.format_text(text: vvaluex(ItemText), overrides: TextOverrides);
      end;
  %mend;
```

If the value of ItemName was found, the XPos and YPos values (from the spreadsheet) are used in the REGION method call to set the position on the page where the next output will be placed. The value of ItemText (from the spreadsheet) will be displayed. Also in the spreadsheet are fontsize and fontweight values that are used to build a list of overrides to the default display attributes.

In this example, when PatientName is passed to the macro, it is found in the spreadsheet with the FIND method on the hash object and the associated variables are loaded. So, the X and Y position of the region will 39mm and 22.5mm and the patient's name will print in a 12pt bold font.

In loop that reads the VisitReason reason dataset, the value of the variable VisitReasonCode is looked for in the spreadsheet. A value for symptoms (SYMPT) would print an 8pt bold "X" at 16mm from the left and 103mm down.

This technique allows for printing of almost any type of form. The complete code to create this page is shown in Appendix E-2 and a completed version of the form is shown in Appendix E-3.

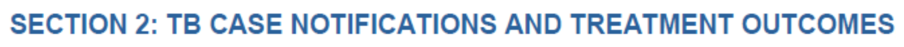
REAL WORLD EXAMPLES: REUSING FEATURES!

As seen with the last example, placing RWI code in a macro can often make the actual data step that creates the report much shorter and easier to read. This final example makes heavy use of macros – in fact, other than the declaration of the ODSOUT object, there is not a single line of RWI code in the data step itself.

The report that is created is for TB case notification for a particular year. Counts of cases are presented in many different ways. The pages contain headers, tables of different structures and simple rows of numbers. Different macros were created for each type of output. I won't present the code here, but will describe what some of the macros do and you can envision how "clean" the data step will look.

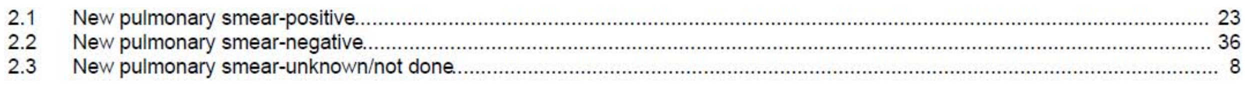
The SectionHeader macro simply uses a FORMAT_TEXT method to place formatted text at the beginning of a section of the report – all that needs to be passed is the text string:

```
%SectionHeader(%str(SECTION 2: TB CASE NOTIFICATIONS AND TREATMENT OUTCOMES))
```


SECTION 2: TB CASE NOTIFICATIONS AND TREATMENT OUTCOMES

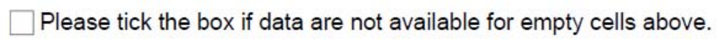
The SimpleQuestion macro takes three parameters: the question number on the form (just text), the question text and the variable that contains the count for that question. This macro uses code very similar to the table of contents example above:

```
%SimpleQuestion(2.1,New pulmonary smear-positive,Q2_01);
%SimpleQuestion(2.2,New pulmonary smear-negative,Q2_02);
%SimpleQuestion(2.3,New pulmonary smear-unknown/not done,Q2_03);
```


```
2.1   New pulmonary smear-positive.................................... 23
2.2   New pulmonary smear-negative.................................... 36
2.3   New pulmonary smear-unknown/not done............................  8
```

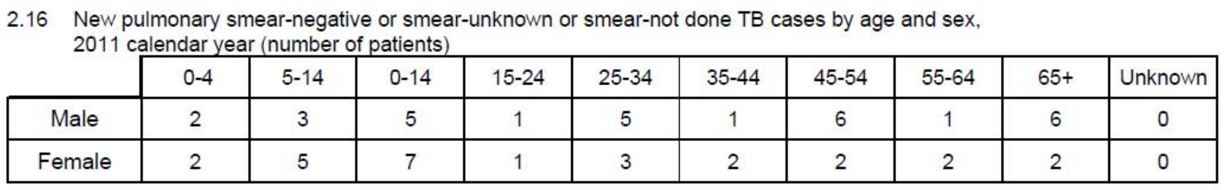
This report also mimics a paper-based report that the user can fill out. For this reason, some of the elements on the report are there for compatibility reasons only. On the paper form, the user has a number of places where they can check if data was not available for a section. The EmptyCellCheckBox macro is just a cell in a table with the text shown below – no parameters need to be passed as, in this report, the text is always the same.

```
%EmptyCellCheckBox;
```


☐ Please tick the box if data are not available for empty cells above.

There are a number of tabular presentations of data in this report. One that is used a number of times is a breakdown of different types of cases by age and gender. The AgeGenderTable macro uses the TABLE and ROW methods (START/END) and the FORMAT_CELL methods to create the tables. The question number and text are passed, similar to the SimpleQuestion macro, along with an array reference, which contains the numbers for the table. Notice that we're taking advantage of the BORDER…COLOR attributes to "turn off" the borders of the upper left cell.

```
%AgeGenderTable(2.16,%str(New pulmonary smear-negative or smear-unknown...,WHO_PSN);
```


2.16 New pulmonary smear-negative or smear-unknown or smear-not done TB cases by age and sex, 2011 calendar year (number of patients)

	0-4	5-14	0-14	15-24	25-34	35-44	45-54	55-64	65+	Unknown
Male	2	3	5	1	5	1	6	1	6	0
Female	2	5	7	1	3	2	2	2	2	0

There are other table-generation macros that create tables with spanning rows, spanning columns, grayed-out cells when data was unavailable, etc. As we've seen in the examples so far, there's not much that cannot be done with a little imagination.

The actual report code contains 14 macros that create output types like those shown above. The single data step generates a four page report with using over 70 calls to those macros. Hopefully, this will give you the idea that RWI code is perfect for a "modularized" implementation. There are often reports that might use the same type of output over and over and this is so easily done with this method.

The first two pages of this report are shown in Appendix F, showing much of what's been discussed here.

CONCLUSION

The Report Writing Interface is a powerful tool in the SAS reporting toolbox. Even though it's still "pre-production" even in v9.3, it has proven to be stable and reliable for many tasks. Take a look at other papers on the SAS Global Forum proceedings web site (see references) and glean what you can. You'll find that you might often turn to the data step rather than a procedure when it's time for the next reporting task.

REFERENCES

AskHYS – a website for reporting of information from the Washington State Healthy Youth Survey: www.AskHYS.net – click on HYS Results…Frequency Reports. Then, select 2012 and any report from the ESD or County list. These reports were all 100% SAS-generated and make extensive use of the Report Writing Interface.

Lund, Pete, “You Did That Report in SAS®!?: The Power of the ODS PDF Destination,” *Proceedings of the 2011 SAS Global Forum Conference*, SAS Institute Inc. (Cary, NC), 2011. (http://support.sas.com/resources/papers/proceedings11/247-2011.pdf)

O’Connor, Daniel, “The Power to Show: Ad Hoc Reporting, Custom Invoices, and Form Letters,” *Proceedings of the 2009 SAS Global Forum Conference*, SAS Institute Inc. (Cary, NC), 2009. (http://support.sas.com/rnd/base/datastep/dsobject/Power_to_show_paper.pdf - this is an updated version of the paper presented at the conference)

AUTHOR CONTACT INFORMATION

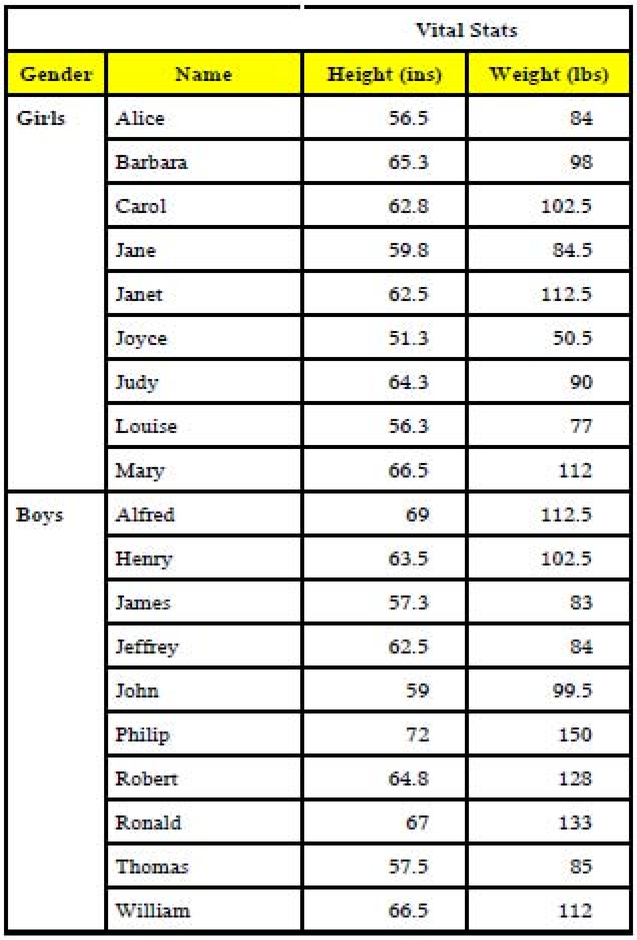
Pete Lund
Looking Glass Analytics
215 Legion Way SW
Olympia, WA 98501
(360) 528-8970
pete.lund@lgan.com

ACKNOWLEDGEMENTS

SAS® is a registered trademark of SAS Institute, Inc. in the USA and other countries. Other products are registered trademarks or trademarks of their respective companies.

Appendix A
Example of Basic Table Creation with the Report Writing Interface

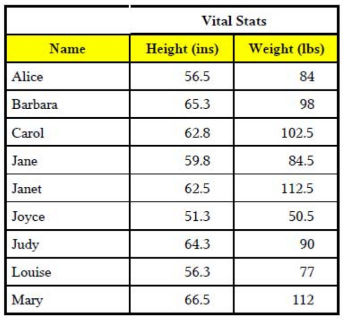
```
proc template;
  define style test;
  parent=styles.printer;

  style datacells from body /
    just=right
    rightmargin=4mm;
  style headercells from body /
    background=yellow fontweight=bold;
  end;
run;

proc sql;
  create table numsex as
  select sex,count(*) as num
  from sashelp.class
  group by 1;

  create table newclass as
  select c.*,
         num
  from sashelp.class c,
       numsex n
  where c.sex eq n.sex
  order by sex,name;
quit;

proc format;
  value $Sex
    'F' = 'Girls'
    'M' = 'Boys';
run;

ods pdf notoc style=test;

data _null_;
  set newclass end=done;
  by sex;

  if _n_ eq 1 then
    do;
      declare odsout t();

      t.table_start();
        t.row_start();
          t.format_cell(colspan: 2);
          t.format_cell(text: 'Vital Stats',
                        overrides: 'borderleftcolor=white fontweight=bold', colspan: 2);
        t.row_end();


      t.row_start();
        t.format_cell(text: 'Gender',style: 'HeaderCells', overrides: 'cellwidth=15mm');
        t.format_cell(text: 'Name',style: 'HeaderCells', overrides: 'cellwidth=30mm');
        t.format_cell(text: 'Height (ins)',style: 'HeaderCells', overrides: 'cellwidth=25mm');
        t.format_cell(text: 'Weight (lbs)',style: 'HeaderCells', overrides: 'cellwidth=25mm');
      t.row_end();
    end;

    t.row_start();
      if first.sex then t.format_cell(text: put(Sex,$Sex.),
                                      overrides:'just=left fontweight=bold vjust=top', rowspan: num);
      t.format_cell(text: name,overrides: 'just=left');
      t.format_cell(text: height, style: 'DataCells');
      t.format_cell(text: weight, style: 'DataCells');
    t.row_end();

  if done then t.table_end();
run;

ods _all_ close;
```



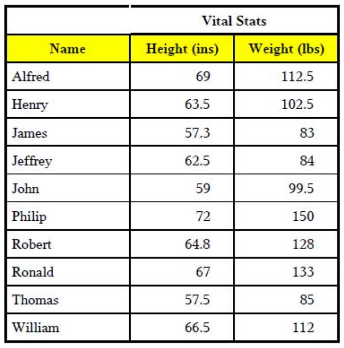
		Vital Stats	
Gender	**Name**	**Height (ins)**	**Weight (lbs)**
Girls	Alice	56.5	84
	Barbara	65.3	98
	Carol	62.8	102.5
	Jane	59.8	84.5
	Janet	62.5	112.5
	Joyce	51.3	50.5
	Judy	64.3	90
	Louise	56.3	77
	Mary	66.5	112
Boys	Alfred	69	112.5
	Henry	63.5	102.5
	James	57.3	83
	Jeffrey	62.5	84
	John	59	99.5
	Philip	72	150
	Robert	64.8	128
	Ronald	67	133
	Thomas	57.5	85
	William	66.5	112

Appendix B
Using the LAYOUT_ABSOLUTE Method for Side-by-Side Output

```
proc template;
  define style test;
  parent=styles.printer;

  style datacells from body /
    just=right
    rightmargin=4mm;
  style headercells from body /
    background=yellow fontweight=bold;
  end;
run;

ods _all_ close;
ods pdf notoc style=test;

data _null_;
  set sortedclass end=done;
  by sex;

  retain Group 0;

  if _n_ eq 1 then
    do;
      declare odsout t();

      t.layout_absolute();
    end;

  if first.sex then
    do;
      t.region(x: catt(Group*4,'in'), y: '1in', width: '3.5in');

      Group + 1;

      t.table_start();
        t.row_start();
          t.format_cell();
          t.format_cell(text: 'Vital Stats', overrides: 'borderleftcolor=white fontweight=bold',
                        colspan: 2);
        t.row_end();

      t.row_start();
        t.format_cell(text: 'Name',style: 'HeaderCells', overrides: 'cellwidth=30mm');
        t.format_cell(text: 'Height (ins)',style: 'HeaderCells', overrides: 'cellwidth=25mm');
        t.format_cell(text: 'Weight (lbs)',style: 'HeaderCells', overrides: 'cellwidth=25mm');
      t.row_end();
    end;

    t.row_start();
      t.format_cell(text: name,overrides: 'just=left');
      t.format_cell(text: height, style: 'DataCells');
      t.format_cell(text: weight, style: 'DataCells');
    t.row_end();

  if last.sex then t.table_end();
  if done then t.layout_end();
run;

ods _all_ close;
```

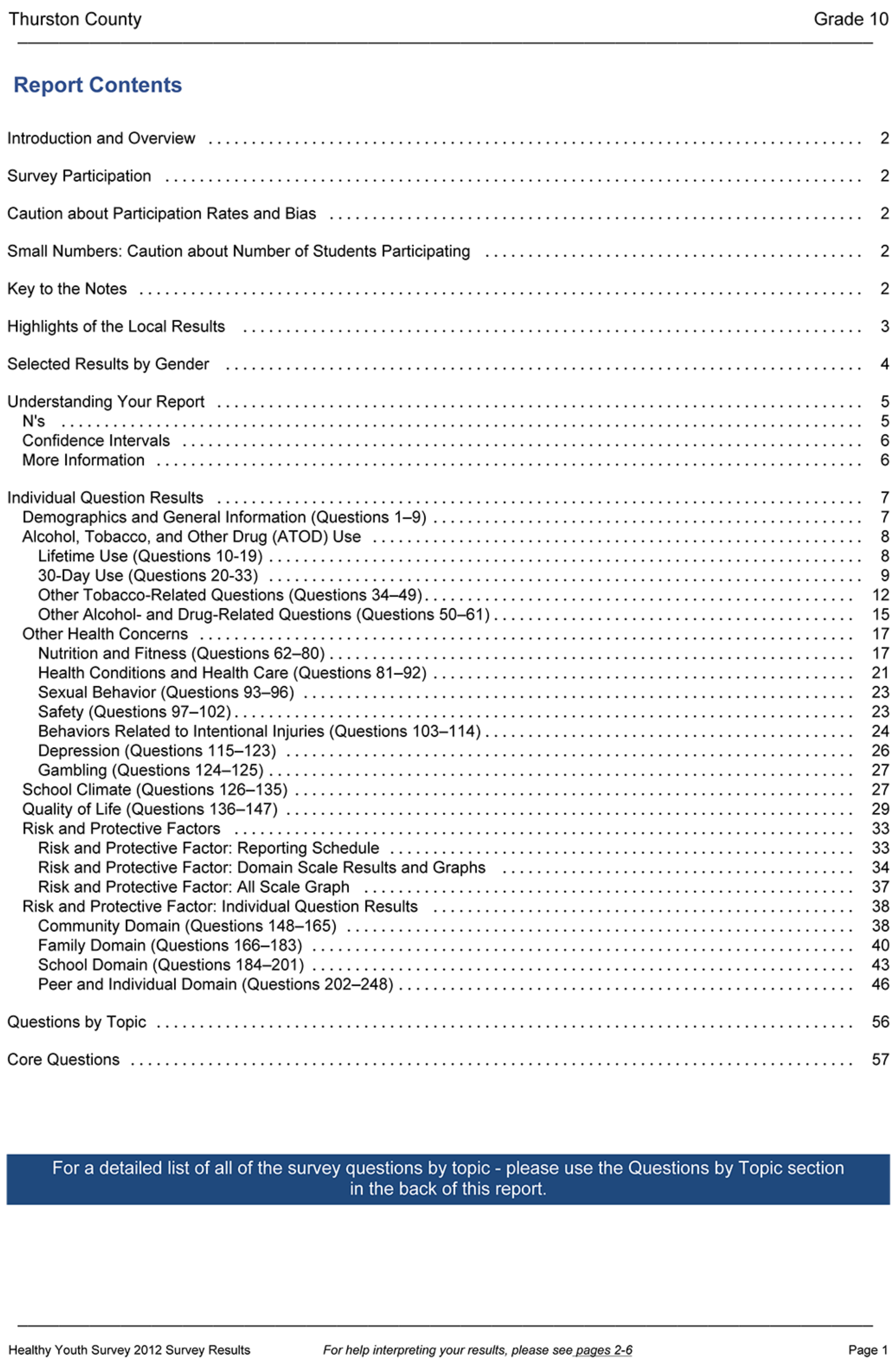


	Vital Stats	
Name	Height (ins)	Weight (lbs)
Alice	56.5	84
Barbara	65.3	98
Carol	62.8	102.5
Jane	59.8	84.5
Janet	62.5	112.5
Joyce	51.3	50.5
Judy	64.3	90
Louise	56.3	77
Mary	66.5	112



	Vital Stats	
Name	Height (ins)	Weight (lbs)
Alfred	69	112.5
Henry	63.5	102.5
James	57.3	83
Jeffrey	62.5	84
John	59	99.5
Philip	72	150
Robert	64.8	128
Ronald	67	133
Thomas	57.5	85
William	66.5	112

Appendix C
A Simple Table of Contents Created with the Report Writing Interface



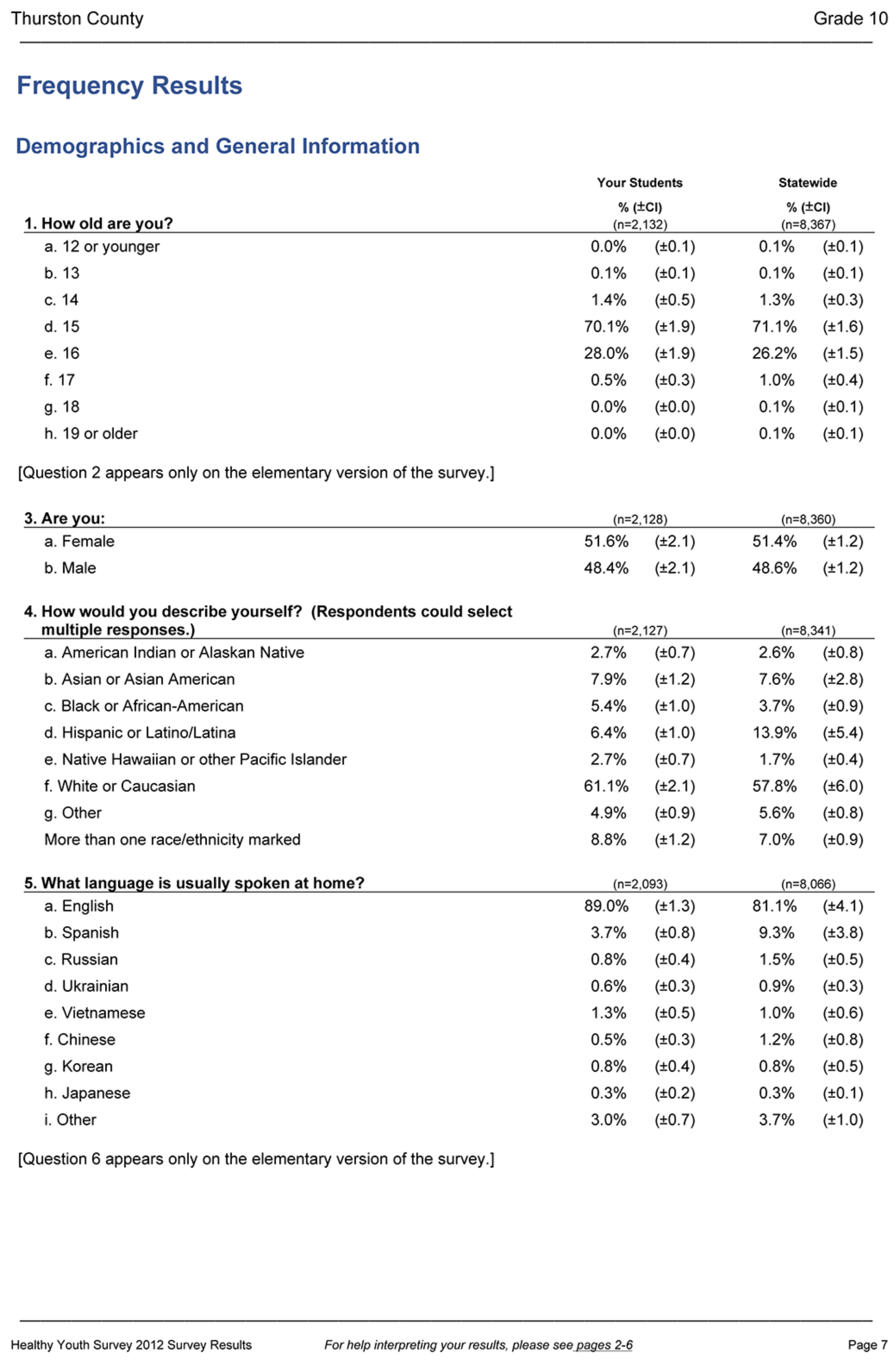
Thurston County Grade 10

Report Contents

For a detailed list of all of the survey questions by topic - please use the Questions by Topic section in the back of this report.

Healthy Youth Survey 2012 Survey Results *For help interpreting your results, please see pages 2-6* Page 1

Appendix D
“Replacing PROC REPORT” with the Report Writing Interface



Thurston County Grade 10

Frequency Results

Demographics and General Information

1. How old are you?	**Your Students** % (±CI) (n=2,132)		**Statewide** % (±CI) (n=8,367)	
a. 12 or younger	0.0%	(±0.1)	0.1%	(±0.1)
b. 13	0.1%	(±0.1)	0.1%	(±0.1)
c. 14	1.4%	(±0.5)	1.3%	(±0.3)
d. 15	70.1%	(±1.9)	71.1%	(±1.6)
e. 16	28.0%	(±1.9)	26.2%	(±1.5)
f. 17	0.5%	(±0.3)	1.0%	(±0.4)
g. 18	0.0%	(±0.0)	0.1%	(±0.1)
h. 19 or older	0.0%	(±0.0)	0.1%	(±0.1)

[Question 2 appears only on the elementary version of the survey.]

3. Are you:	(n=2,128)		(n=8,360)	
a. Female	51.6%	(±2.1)	51.4%	(±1.2)
b. Male	48.4%	(±2.1)	48.6%	(±1.2)

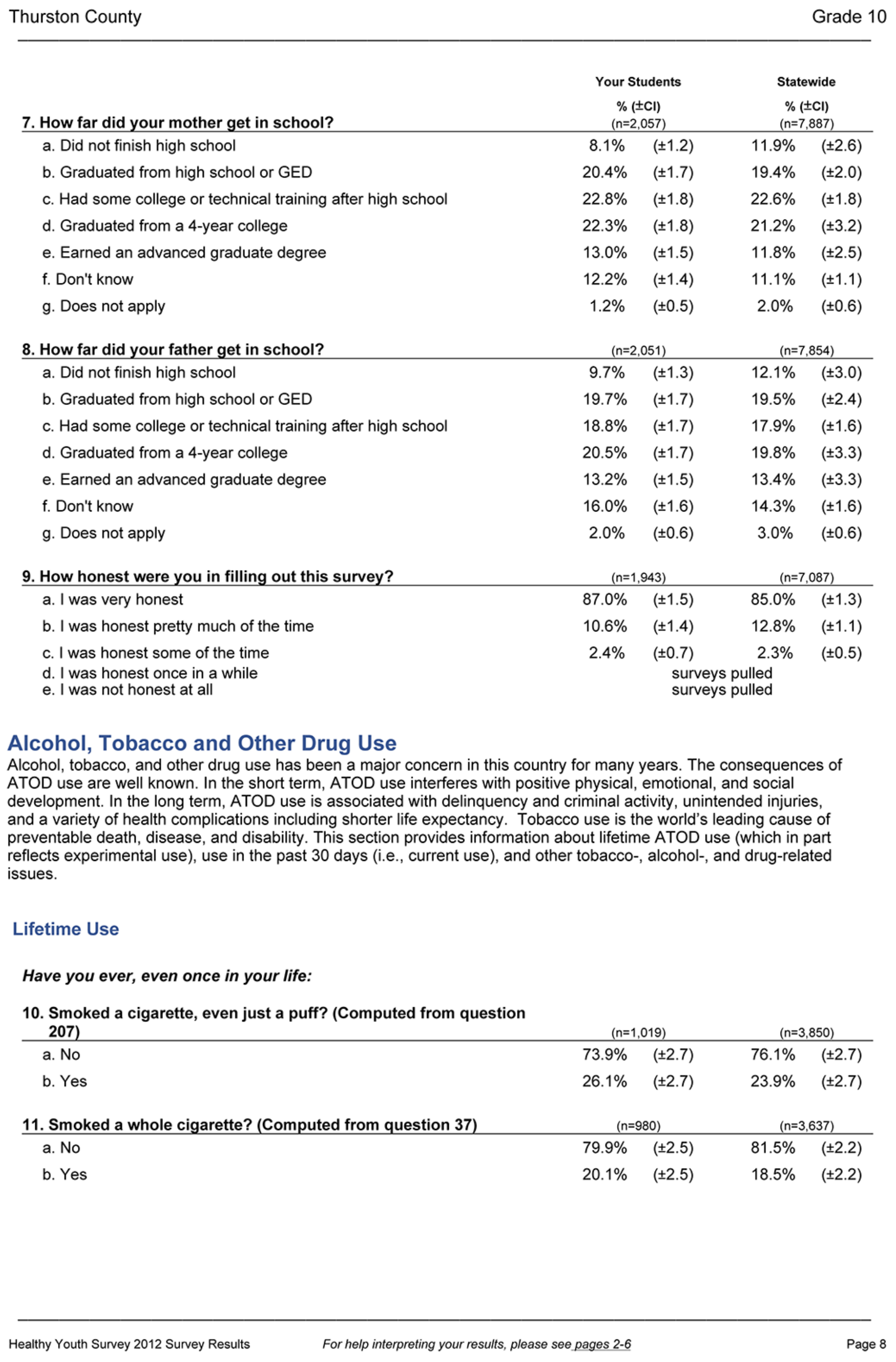
4. How would you describe yourself? (Respondents could select multiple responses.)	(n=2,127)		(n=8,341)	
a. American Indian or Alaskan Native	2.7%	(±0.7)	2.6%	(±0.8)
b. Asian or Asian American	7.9%	(±1.2)	7.6%	(±2.8)
c. Black or African-American	5.4%	(±1.0)	3.7%	(±0.9)
d. Hispanic or Latino/Latina	6.4%	(±1.0)	13.9%	(±5.4)
e. Native Hawaiian or other Pacific Islander	2.7%	(±0.7)	1.7%	(±0.4)
f. White or Caucasian	61.1%	(±2.1)	57.8%	(±6.0)
g. Other	4.9%	(±0.9)	5.6%	(±0.8)
More than one race/ethnicity marked	8.8%	(±1.2)	7.0%	(±0.9)

5. What language is usually spoken at home?	(n=2,093)		(n=8,066)	
a. English	89.0%	(±1.3)	81.1%	(±4.1)
b. Spanish	3.7%	(±0.8)	9.3%	(±3.8)
c. Russian	0.8%	(±0.4)	1.5%	(±0.5)
d. Ukrainian	0.6%	(±0.3)	0.9%	(±0.3)
e. Vietnamese	1.3%	(±0.5)	1.0%	(±0.6)
f. Chinese	0.5%	(±0.3)	1.2%	(±0.8)
g. Korean	0.8%	(±0.4)	0.8%	(±0.5)
h. Japanese	0.3%	(±0.2)	0.3%	(±0.1)
i. Other	3.0%	(±0.7)	3.7%	(±1.0)

[Question 6 appears only on the elementary version of the survey.]

Healthy Youth Survey 2012 Survey Results *For help interpreting your results, please see pages 2-6* Page 7

Appendix D (cont)
"Replacing PROC REPORT" with the Report Writing Interface



Thurston County Grade 10

	Your Students		Statewide	
	% (±CI)		% (±CI)	
7. How far did your mother get in school?	(n=2,057)		(n=7,887)	
a. Did not finish high school	8.1%	(±1.2)	11.9%	(±2.6)
b. Graduated from high school or GED	20.4%	(±1.7)	19.4%	(±2.0)
c. Had some college or technical training after high school	22.8%	(±1.8)	22.6%	(±1.8)
d. Graduated from a 4-year college	22.3%	(±1.8)	21.2%	(±3.2)
e. Earned an advanced graduate degree	13.0%	(±1.5)	11.8%	(±2.5)
f. Don't know	12.2%	(±1.4)	11.1%	(±1.1)
g. Does not apply	1.2%	(±0.5)	2.0%	(±0.6)
8. How far did your father get in school?	(n=2,051)		(n=7,854)	
a. Did not finish high school	9.7%	(±1.3)	12.1%	(±3.0)
b. Graduated from high school or GED	19.7%	(±1.7)	19.5%	(±2.4)
c. Had some college or technical training after high school	18.8%	(±1.7)	17.9%	(±1.6)
d. Graduated from a 4-year college	20.5%	(±1.7)	19.8%	(±3.3)
e. Earned an advanced graduate degree	13.2%	(±1.5)	13.4%	(±3.3)
f. Don't know	16.0%	(±1.6)	14.3%	(±1.6)
g. Does not apply	2.0%	(±0.6)	3.0%	(±0.6)
9. How honest were you in filling out this survey?	(n=1,943)		(n=7,087)	
a. I was very honest	87.0%	(±1.5)	85.0%	(±1.3)
b. I was honest pretty much of the time	10.6%	(±1.4)	12.8%	(±1.1)
c. I was honest some of the time	2.4%	(±0.7)	2.3%	(±0.5)
d. I was honest once in a while		surveys pulled		
e. I was not honest at all		surveys pulled		

Alcohol, Tobacco and Other Drug Use

Alcohol, tobacco, and other drug use has been a major concern in this country for many years. The consequences of ATOD use are well known. In the short term, ATOD use interferes with positive physical, emotional, and social development. In the long term, ATOD use is associated with delinquency and criminal activity, unintended injuries, and a variety of health complications including shorter life expectancy. Tobacco use is the world's leading cause of preventable death, disease, and disability. This section provides information about lifetime ATOD use (which in part reflects experimental use), use in the past 30 days (i.e., current use), and other tobacco-, alcohol-, and drug-related issues.

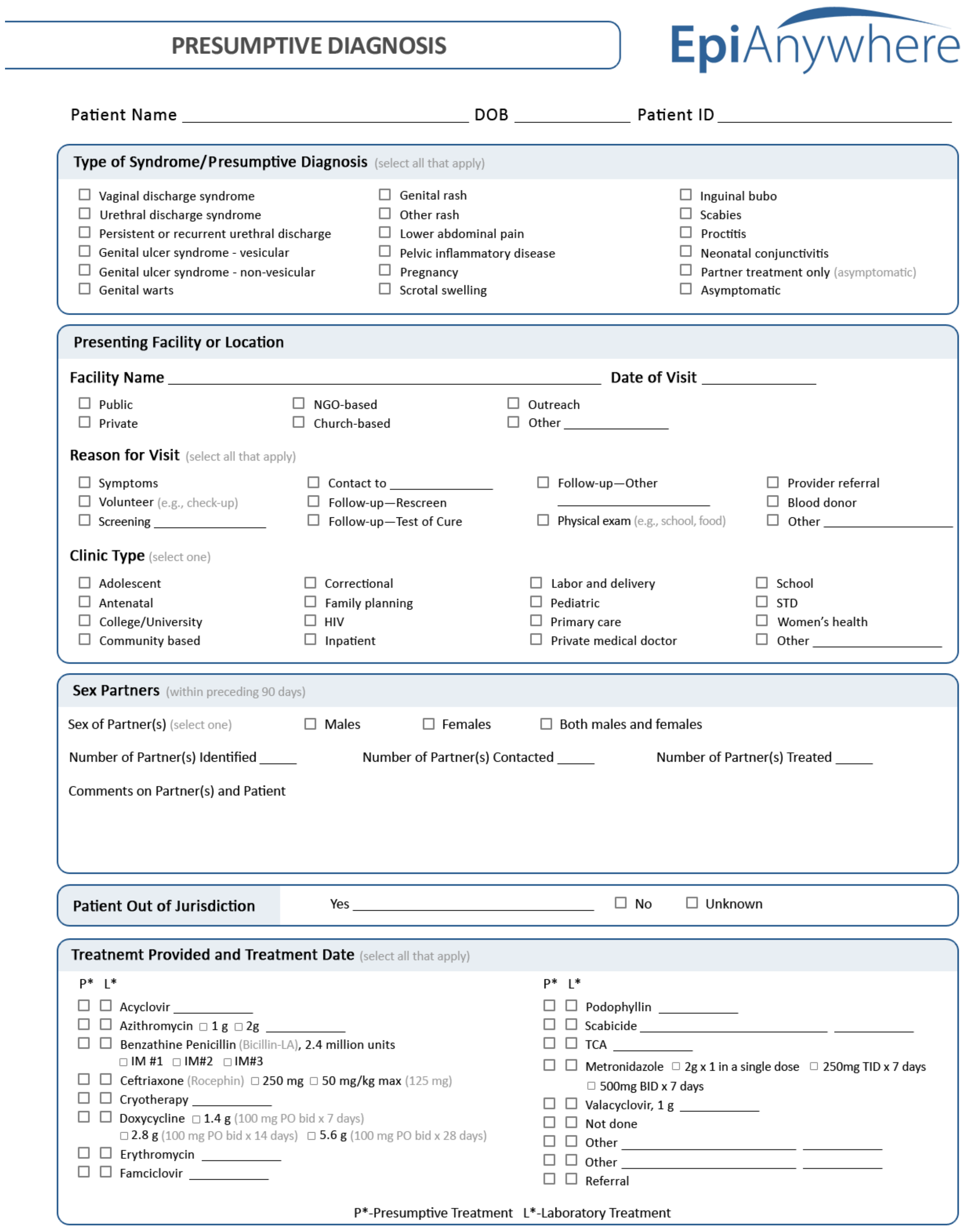
Lifetime Use

Have you ever, even once in your life:

10. Smoked a cigarette, even just a puff? (Computed from question 207)	(n=1,019)		(n=3,850)	
a. No	73.9%	(±2.7)	76.1%	(±2.7)
b. Yes	26.1%	(±2.7)	23.9%	(±2.7)
11. Smoked a whole cigarette? (Computed from question 37)	(n=980)		(n=3,637)	
a. No	79.9%	(±2.5)	81.5%	(±2.2)
b. Yes	20.1%	(±2.5)	18.5%	(±2.2)

Healthy Youth Survey 2012 Survey Results *For help interpreting your results, please see pages 2-6* Page 8

Appendix E-1
Using LAYOUT_ABSOLUTE to Fill a Form
The Empty Form



PRESUMPTIVE DIAGNOSIS

EpiAnywhere

Patient Name ______________________________ DOB ____________ Patient ID _________________________

Type of Syndrome/Presumptive Diagnosis (select all that apply)

☐ Vaginal discharge syndrome	☐ Genital rash	☐ Inguinal bubo
☐ Urethral discharge syndrome	☐ Other rash	☐ Scabies
☐ Persistent or recurrent urethral discharge	☐ Lower abdominal pain	☐ Proctitis
☐ Genital ulcer syndrome - vesicular	☐ Pelvic inflammatory disease	☐ Neonatal conjunctivitis
☐ Genital ulcer syndrome - non-vesicular	☐ Pregnancy	☐ Partner treatment only (asymptomatic)
☐ Genital warts	☐ Scrotal swelling	☐ Asymptomatic

Presenting Facility or Location

Facility Name ____ **Date of Visit** ____________

☐ Public	☐ NGO-based	☐ Outreach
☐ Private	☐ Church-based	☐ Other ___________

Reason for Visit (select all that apply)

☐ Symptoms	☐ Contact to ___________	☐ Follow-up—Other	☐ Provider referral
☐ Volunteer (e.g., check-up)	☐ Follow-up—Rescreen	________________	☐ Blood donor
☐ Screening ___________	☐ Follow-up—Test of Cure	☐ Physical exam (e.g., school, food)	☐ Other _____________

Clinic Type (select one)

☐ Adolescent	☐ Correctional	☐ Labor and delivery	☐ School
☐ Antenatal	☐ Family planning	☐ Pediatric	☐ STD
☐ College/University	☐ HIV	☐ Primary care	☐ Women's health
☐ Community based	☐ Inpatient	☐ Private medical doctor	☐ Other _____________

Sex Partners (within preceding 90 days)

Sex of Partner(s) (select one) ☐ Males ☐ Females ☐ Both males and females

Number of Partner(s) Identified _____ Number of Partner(s) Contacted _____ Number of Partner(s) Treated _____

Comments on Partner(s) and Patient

Patient Out of Jurisdiction Yes ___________________________ ☐ No ☐ Unknown

Treatnemt Provided and Treatment Date (select all that apply)

P* L*

☐ ☐ Acyclovir _________

☐ ☐ Azithromycin ☐ 1 g ☐ 2g _________

☐ ☐ Benzathine Penicillin (Bicillin-LA), 2.4 million units
☐ IM #1 ☐ IM#2 ☐ IM#3

☐ ☐ Ceftriaxone (Rocephin) ☐ 250 mg ☐ 50 mg/kg max (125 mg)

☐ ☐ Cryotherapy _________

☐ ☐ Doxycycline ☐ 1.4 g (100 mg PO bid x 7 days)
☐ 2.8 g (100 mg PO bid x 14 days) ☐ 5.6 g (100 mg PO bid x 28 days)

☐ ☐ Erythromycin _________

☐ ☐ Famciclovir _________

P* L*

☐ ☐ Podophyllin _________

☐ ☐ Scabicide ____________________ _________

☐ ☐ TCA

☐ ☐ Metronidazole ☐ 2g x 1 in a single dose ☐ 250mg TID x 7 days
☐ 500mg BID x 7 days

☐ ☐ Valacyclovir, 1 g _________

☐ ☐ Not done

☐ ☐ Other ____________________ _________

☐ ☐ Other ____________________ _________

☐ ☐ Referral

P*-Presumptive Treatment L*-Laboratory Treatment

Appendix E-2
Using LAYOUT_ABSOLUTE to Fill a Form
Complete Code to Fill the Form

```
proc template;
  define style work.sti_journal;
  parent=styles.journal;
  style TestInfoHeader from document /
    font_face=Helvetica
    fontsize=7pt
    fontweight=bold
    background=cxFFFFFF
    just=center;
  style TestInfo from document /
    font_face=Helvetica
    fontsize=7pt
    just=left;
  end;

  <page 1 definition>

  define style work.myjournal_p2;
  parent=work.sti_journal;
  style body / backgroundimage="&ImagePath\STI_form_FINAL-2.png?width=100%nrstr(%&)height=100%nrstr(%%)"
   margintop=0mm marginleft=0mm height=10.5in width=8in;
  end;

  <page 3 definition>
run;

ods pdf file=<file location> notoc style=work.myjournal_p1;

<page 1 code>

  ods pdf style=work.myjournal_p2;

  data _null_;
    length ItemName $40 ItemText $100 Xpos Ypos FontSize 4 FontWeight $10 TextOverrides $200;
    call missing(ItemName, ItemText, Xpos, Ypos, FontSize, FontWeight);

    retain X 'X';

    if _n_ eq 1 then
      do;
        declare hash pl(dataset: "pl.'PageLayout$'n");
        pl.defineKey("ItemName");
        pl.defineData('Xpos','Ypos','ItemText','FontSize','FontWeight');
        pl.defineDone();

        declare odsout vi();

        vi.layout_absolute();
      end;

    do while (not endPatient);
      set PatientInfo end=endPatient;

      %DisplayInfo(PatientName_2);
      %DisplayInfo(PatientID_2);
      %DisplayInfo(DOB_2);
      %DisplayInfo(DOV);
      %DisplayInfo(FacilityName);
      %DisplayInfo(STIFacilityCode,GetValue=Y);
```

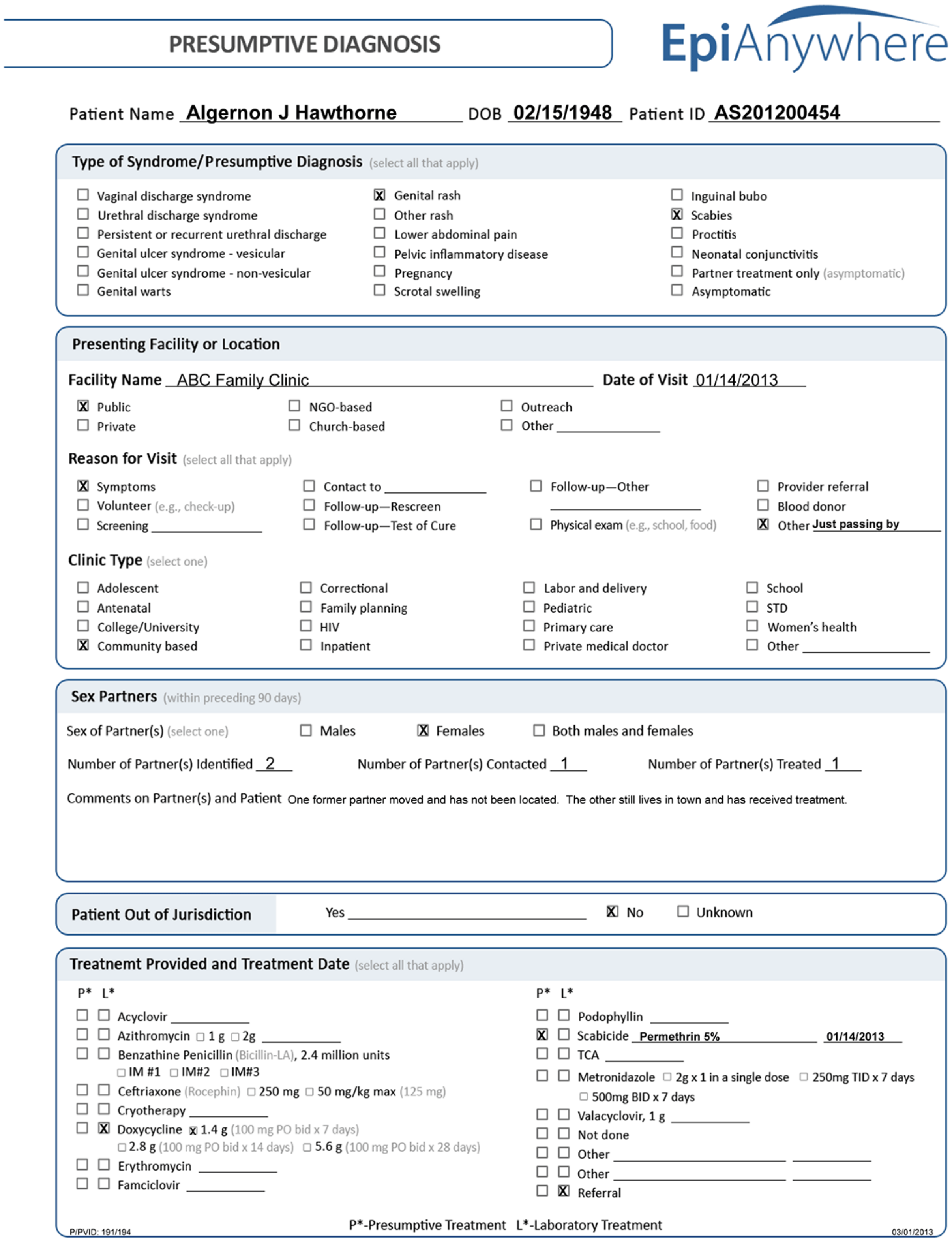
The purpose of the complete code is not to be able to reuse it, but to show that it's not all that difficult to create a pretty complex form, assuming that the data have been processed and the position/style information have been set up correctly.

This is a technique I use frequently and it allows for a SAS-generated form that looks exactly as the user expects.

```
      %DisplayInfo(STIClinicCode,GetValue=Y);
      %DisplayInfo(ClinicOther);
      %DisplayInfo(PartnerGenderCode,GetValue=Y);
      %DisplayInfo(PartnersIdentified);
      %DisplayInfo(PartnersContacted);
      %DisplayInfo(PartnersTreated);
      %DisplayInfo(PartnerComments);
      %DisplayInfo(PatientOutOfJurisdiction);
      %DisplayInfo(PatientOutOfJurisdictionNoUnk,GetValue=Y);
    end;

    do while (not endReason);
      set VisitReason end=endReason;
      %DisplayInfo(STIVisitReasonCode,GetValue=Y);
    end;

    do while (not endReasonOther);
      set VisitReasonOther end=endReasonOther;
      %DisplayInfo(VisitReasonOther,GetValue=Y);
    end;

    do while (not endPD);
      set VisitPD end=endPD;
      %DisplayInfo(STISyndromeCode,GetValue=Y);
    end;

    do while (not endTx);
      set Treatments end=endTx;

      TreatmentCodeMain = compress(STITreatmentCode,'0123456789');
      TreatmentCodeByClass = catx('_',TreatmentCodeMain,TreatmentClass);
      if STITreatmentCode ne TreatmentCodeMain then TreatmentCodeDetail = STITreatmentCode;

      %DisplayInfo(TreatmentCodeByClass,GetValue=Y);

      if STITreatmentCode ne TreatmentCodeMain then
        do;
          %DisplayInfo(STITreatmentCode,GetValue=Y);
        end;

      if TreatmentDate ne . then
        do;
          TxDate = put(TreatmentDate,mmddyy10.);
          DateField = catx('_',TreatmentCodeMain,'Date');
          %DisplayInfo(DateField,GetValue=Y);
        end;

      if STITreatmentSpecify ne '' then
        do;
          DetailField = catx('_',TreatmentCodeMain,'Detail');
          %DisplayInfo(DetailField,GetValue=Y);
        end;
    end;

    %VisitReportInfo;

    vi.layout_end();

    stop;
  run;

<page 3 code>

ods pdf close;
```

Appendix E-3
Using LAYOUT_ABSOLUTE to Fill a Form

The Filled Form



PRESUMPTIVE DIAGNOSIS

EpiAnywhere

Patient Name Algernon J Hawthorne DOB 02/15/1948 Patient ID AS201200454

Type of Syndrome/Presumptive Diagnosis (select all that apply)

☐ Vaginal discharge syndrome	[✓ Genital rash]	☐ Inguinal bubo
☐ Urethral discharge syndrome	☐ Other rash	[✓ Scabies]
☐ Persistent or recurrent urethral discharge	☐ Lower abdominal pain	☐ Proctitis
☐ Genital ulcer syndrome - vesicular	☐ Pelvic inflammatory disease	☐ Neonatal conjunctivitis
☐ Genital ulcer syndrome - non-vesicular	☐ Pregnancy	☐ Partner treatment only (asymptomatic)
☐ Genital warts	☐ Scrotal swelling	☐ Asymptomatic

Presenting Facility or Location

Facility Name ABC Family Clinic **Date of Visit** 01/14/2013

[✓ Public]	☐ NGO-based	☐ Outreach
☐ Private	☐ Church-based	☐ Other ____________

Reason for Visit (select all that apply)

[✓ Symptoms]	☐ Contact to ____________	☐ Follow-up—Other ____________	☐ Provider referral
☐ Volunteer (e.g., check-up)	☐ Follow-up—Rescreen		☐ Blood donor
☐ Screening ____________	☐ Follow-up—Test of Cure	☐ Physical exam (e.g., school, food)	[✓ Other Just passing by]

Clinic Type (select one)

☐ Adolescent	☐ Correctional	☐ Labor and delivery	☐ School
☐ Antenatal	☐ Family planning	☐ Pediatric	☐ STD
☐ College/University	☐ HIV	☐ Primary care	☐ Women's health
[✓ Community based]	☐ Inpatient	☐ Private medical doctor	☐ Other ____________

Sex Partners (within preceding 90 days)

Sex of Partner(s) (select one) ☐ Males [✓ Females] ☐ Both males and females

Number of Partner(s) Identified 2 Number of Partner(s) Contacted 1 Number of Partner(s) Treated 1

Comments on Partner(s) and Patient One former partner moved and has not been located. The other still lives in town and has received treatment.

Patient Out of Jurisdiction Yes ____________ [✓ No] ☐ Unknown

Treatemt Provided and Treatment Date (select all that apply)

P*	L*		P*	L*	
☐	☐	Acyclovir ________	☐	☐	Podophyllin ________
☐	☐	Azithromycin ☐ 1 g ☐ 2g ________	[✓]	☐	Scabicide Permethrin 5% 01/14/2013
☐	☐	Benzathine Penicillin (Bicillin-LA), 2.4 million units ☐ IM #1 ☐ IM#2 ☐ IM#3	☐	☐	TCA ________
☐	☐	Ceftriaxone (Rocephin) ☐ 250 mg ☐ 50 mg/kg max (125 mg)	☐	☐	Metronidazole ☐ 2g x 1 in a single dose ☐ 250mg TID x 7 days ☐ 500mg BID x 7 days
☐	☐	Cryotherapy ________	☐	☐	Valacyclovir, 1 g ________
☐	[✓]	Doxycycline [✓ 1.4 g] (100 mg PO bid x 7 days) ☐ 2.8 g (100 mg PO bid x 14 days) ☐ 5.6 g (100 mg PO bid x 28 days)	☐	☐	Not done
☐	☐	Erythromycin ________	☐	☐	Other ________ ________
☐	☐	Famciclovir ________	☐	☐	Other ________ ________
			☐	[✓]	Referral

P/PVID: 191/194 P*-Presumptive Treatment L*-Laboratory Treatment 03/01/2013

Appendix F
Using the Report Writing Interface to Create Complex Output

SECTION 2: TB CASE NOTIFICATIONS AND TREATMENT OUTCOMES, Guam, 2011 cases

TB cases by history, site and smear results, 2011 calendar year (number of patients)

2.1	New pulmonary smear-positive	23
2.2	New pulmonary smear-negative	36
2.3	New pulmonary smear-unknown/not done	8
2.4	New extrapulmonary	8
2.5	Other NEW cases not in lines 2.1-2.4	0
2.6	Relapse (pulmonary smear and/or culture-positive)	4
2.7	Treatment after failure (pulmonary smear and/or culture-positive)	0
2.8	Treatment after default (pulmonary smear and/or culture-positive)	0
2.9	Other re-treatment cases not in lines 2.6-2.8	0
2.10	Other, not in lines 2.1-2.9 (i.e., history unknown)	1
2.11	New pulmonary laboratory-confirmed cases. Laboratory-confirmed includes all cases confirmed by smear and/or culture, or by any other laboratory methods	41
2.12	Total number of new and re-treatment TB cases reported among foreign-born individuals (or among non-citizens, if that is the criterion used in your country)	50
2.13	Number of people with signs and symptoms suggestive of pulmonary TB (e.g. cough of long duration, more than 2-3 weeks) screened for TB	n/a
2.14	Number of TB deaths registered by the vital registration system of your country following the ICD-10 (or ICD-9) codes for TB	4

☐ Please tick the box if data are not available for empty cells above.

2.15 New pulmonary smear-positive TB cases by age and sex, 2011 calendar year (number of patients)

	0-4	5-14	0-14	15-24	25-34	35-44	45-54	55-64	65+	Unknown
Male	0	0	0	1	0	2	5	4	3	0
Female	0	0	0	1	1	1	0	2	3	0

☐ Please tick the box if data are not available for empty cells above.

2.16 New pulmonary smear-negative or smear-unknown or smear-not done TB cases by age and sex, 2011 calendar year (number of patients)

	0-4	5-14	0-14	15-24	25-34	35-44	45-54	55-64	65+	Unknown
Male	2	3	5	1	5	1	6	1	6	0
Female	2	5	7	1	3	2	2	2	2	0

☐ Please tick the box if data are not available for empty cells above.

2.17 New extrapulmonary TB cases by age and sex, 2011 calendar year (number of patients)

	0-4	5-14	0-14	15-24	25-34	35-44	45-54	55-64	65+	Unknown
Male	0	0	0	1	0	0	1	0	2	0
Female	0	0	0	1	0	0	0	3	0	0

☐ Please tick the box if data are not available for empty cells above.

MDR-TB, 2011 calendar year (number of patients)

2.18	Number of labratory-confirmed MDR-TB cases identified among all TB patients (new, previously treated, or unknown treatment history) in 2011	0

		2010	2011
2.19	Number of MDR-TB patients who started treatment in GLC-approved projects	0	0
2.20	Number of MDR-TB patients who started treatment outside GLC-approved projects	2	0

☐ Please tick the box if data are not available for empty cells above.

Note: Data to answer question 2.21 is not available in the EpiAnywhere application

Appendix F (cont)
Using the Report Writing Interface to Create Complex Output

MDR-TB, 2011 calendar year (number of patients), continued

2.22 Results of first-line drug susceptibility testing

	Previous anti-TB treatment Status		
	New	Previously treated	Unknown treatment history
(i) Number of patients with positive identification for *M. Tuberculosis* complex (confirmed by culture and/or line-probe assay)	37	2	0
(ii) Among patients reported in (i), number of patients with DST results for isoniazid (H) and rifampicin (R)	36	2	0
(iii) Among patients reported in (ii), number of patients with resistance to H but not R	4	0	0
(iv) Among patients reported in (ii), number of patients with resistance to R but not H	0	0	0
(v) Among patients reported in (ii), number of patients with resistance to H and R (MDR-TB)	0	0	0

2.23 Association between MDR-TB and HIV status (number of patients)

	HIV Status		
	+	-	Unknown
MDR-TB (resistant to both H and R)	0	0	0
Not MDR-TB (drug susceptible plus any resistance that is not MDR-TB)	0	27	11

2.24 Association between MDR-TB and sex (number of patients)

	Sex		
	Female	Male	Unknown
MDR-TB (resistant to both H and R)	0	0	0
Not MDR-TB (drug susceptible plus any resistance that is not MDR-TB)	13	25	0

2.25 Results of second-line drug susceptibility testing

	Previous Anti-TB treatment status		
	New	Previously treated	Unknown treatment history
(i) Total number of MDR-TB patients with DST results for any flouroquinolone (FQ) and any second-line injectable agent (2LI)	0	0	0
(ii) Among MDR-TB patients reported in (i), number of patients susceptible to both FQ and 2LI	0	0	0
(iii) Among MDR-TB patients reported in (i), number of patients with any resistance to FQ	0	0	0
(iv) Among MDR-TB patients reported in (i), number of patients with any resistance to 2LI	0	0	0
(v) Among MDR-TB patients reported in (i), number of patients with any resistance to both FQ and 2LI (XDR-TB)	0	0	0

☐ Please tick the box if data are not available for empty cells above.