Reporting Macroset

Pascal Hanssens, Business & Decision Life Sciences, Brussels, Belgium

INTRODUCTION

As CRO, we cater to the needs of many companies, many of which have different likes and needs as well as different standards for what they want their target data reports to look like. Companies are used to seeing reports in their standard, and therefore tend to not outsource the reporting. Many already have systems in place to pour their data into the company's fixed structure.

It is difficult for a CRO to have a fixed structure like this. We are presented with the difficulty of having to be able to use any data, and apply any report structure to it. A solution had to be found; This solution, on the front-end side, would be to use templates. A dynamic template, if implemented correctly, can display any report structure the client wishes. All it takes is a SAS® program in the backend that can handle the interpretation of the template.

REPORT EXAMPLE

[redacted] - DSMB - CUT-OFF DATE: 23APR2010 - DUMMY 04MAY2010 10:58:29 - Page 1 of 72

DISPLAY SAF.2 ADVERSE EVENTS: TABULATION OF ALL EVENTS
ITT POPULATION

PHASE = [redacted]

BODY SYSTEM OR ORGAN CLASS / PREFERRED TERM	A n=65 N (%)	B n=68 N (%)	C n=65 N (%)	D n=67 N (%)	E n=64 N (%)	F n=64 N (%)	G n=64 N (%)	OVERALL n=457 N (%)
SUBJECTS HAVING ANY EVENT	60 (92.3%)	61 (89.7%)	61 (93.8%)	64 (95.5%)	58 (90.6%)	59 (92.2%)	55 (85.9%)	418 (91.5%)
BLOOD AND LYMPHATIC SYSTEM DISORDERS	13 (20.0%)	11 (16.2%)	12 (18.5%)	22 (32.8%)	15 (23.4%)	17 (26.6%)	7 (10.9%)	97 (21.2%)
ANAEMIA	5 (7.7%)	2 (2.9%)	3 (4.6%)	8 (11.9%)	10 (15.6%)	5 (7.8%)	2 (3.1%)	35 (7.7%)
LEUKOPENIA	1 (1.5%)	2 (2.9%)	3 (4.6%)	3 (4.5%)	2 (3.1%)	1 (1.6%)	1 (1.6%)	13 (2.8%)
LYMPHOPENIA	0	0	0	1 (1.5%)	0	1 (1.6%)	0	2 (0.4%)
NEUTROPENIA	6 (9.2%)	8 (11.8%)	9 (13.8%)	15 (22.4%)	8 (12.5%)	15 (23.4%)	3 (4.7%)	64 (14.0%)
PANCYTOPENIA	0	0	0	1 (1.5%)	0	0	0	1 (0.2%)
THROMBOCYTOPENIA	2 (3.1%)	0	2 (3.1%)	2 (3.0%)	0	1 (1.6%)	1 (1.6%)	8 (1.8%)
CARDIAC DISORDERS	2 (3.1%)	0	1 (1.5%)	3 (4.5%)	2 (3.1%)	3 (4.7%)	2 (3.1%)	13 (2.8%)
ANGINA PECTORIS	0	0	0	0	0	0	1 (1.6%)	1 (0.2%)
BUNDLE BRANCH BLOCK LEFT	0	0	0	0	1 (1.6%)	0	0	1 (0.2%)
PALPITATIONS	1 (1.5%)	0	0	2 (3.0%)	1 (1.6%)	0	1 (1.6%)	5 (1.1%)
SUPRAVENTRICULAR EXTRASYSTOLES	0	0	1 (1.5%)	0	0	0	0	1 (0.2%)
TACHYCARDIA	1 (1.5%)	0	0	1 (1.5%)	0	2 (3.1%)	1 (1.6%)	5 (1.1%)
VENTRICULAR EXTRASYSTOLES	0	0	0	0	0	1 (1.6%)	0	1 (0.2%)
CONGENITAL, FAMILIAL AND GENETIC DISORDERS	0	0	0	1 (1.5%)	0	0	0	1 (0.2%)
GILBERT'S SYNDROME	0	0	0	1 (1.5%)	0	0	0	1 (0.2%)

N = NUMBER OF SUBJECTS WITH THAT PARTICULAR AE.
% = PERCENTAGE OF SUBJECTS WITH THAT PARTICULAR AE, COMPUTED AGAINST THE TOTAL NUMBER OF SUBJECTS WHO ARE STILL IN THE TRIAL IN THAT PHASE.
FOR AE= THAT HAVE NOT BEEN CODED, THE BODY SYSTEM IS CONSIDERED UNKNOWN AND THE VERBATIM TERM IS SHOWN INSTEAD OF THE PREFERRED TERM.

REPORT POSSIBILITIES

Clients may have different requirements, based on what their intended goal for the report is, but there are some recurring requests:

- Report titles/footers/header: Any report will almost always have these. Implementing a solution will require the template to be able to handle this.
- Denominator information.
- Both dynamic and hardcoded data: Just like you can not have a completely hardcoded report (see the "PREFERRED TERM" label in template #2), in most cases you do need to be able to add hardcoded values.
- By-variables.
- Statistics and summary statistics. The possibility of displaying both base and summary level data (like grouped AE PTs per SOC, see example #2), without having a set input dataset structure.
- Being able to split information into two pages (i.e when treatments don't fit on a single page, while retaining some of the information (see example #2 where overall treatment is repeated on split pages).

The template above allows the user to manually adjust the position of values on the report too. If a client really wants to have that one variable moved two positions to the right, there is no need anymore to find the corresponding piece of code. You simply need to alter your template to move the variable two positions to the right.

TECHNIQUES

META-PROGRAMMING

Because you don't know in advance what your template is going to look like, your SAS® code needs to be very dynamic as well. Based on a select few rules and restrictions, your code will have to build itself around what you specify in your template. You will need to write code that interprets the template and writes code itself, based on what's inside. That type of coding is called meta-programming. Be cautious when applying this technique, as it can quickly lead to code that is very difficult to understand.

MIMICING HIGHER GENERATION LANGUAGES

SPLIT FUNCTIONALITY

In order for your code to be easily overviewed and maintained, first and foremost, it needs to be documented. Having that as a given, a good coding strategy is to take a few steps back before you start programming and consider what you would like to accomplish. You can then write down your goals and look for functionality topics. Whatever falls under a topic, should go in a separate macro.

Applying this technique allows for very easy isolation of bugs, should they ever occur, and easy customization or altering of functionality later on. You extract the macro, alter it, insert it again, and all the code built around it should remain working like it did before.

CREATING A MAIN CONTROLLER MACRO

Having split your functionalities, the next logical step is to start creating your 'main' macro, which calls on the functionalities you've just created. This 'main' macro helps future programmers who look at your code understand how your process works, while also allowing them to disable some of the sub-processes in your system, for ease of debugging, for example.

```
%macro create_report(   template_path=,
                        input_dataset=,
                        title_dataset=,
                        denominator_dataset=,
                        population=,
                        output=,
                        output_library=,
                        program_name=,
                        program_suffix=,
                        by_var=,
                        grouper_variable=,
                        split=0,
                        pagesize=44,
                        empty_report_string=No observations.,
                        page_spread=1
                    );

%let start_process=%sysfunc(time());

%let grouper_variable_without_byvar=&grouper_variable;
%if %scan(&grouper_variable,1,' ') ne &by_var %then %let grouper_variable=&by_var &grouper_variable;
%if &grouper_variable ne %then %let grouper_cnt=%eval(%sysfunc(count(%cmpres(&grouper_variable),%str( )))+1);
%else %let grouper_cnt=0;
%if &split=0 and &by_var ne %then %let split=1;

%if %sysfunc(fexist(filename)) %then filename template clear;;
%if &template_path ne %then filename template "&template_path";;

%assign_titles_and_footers(&title_dataset,&program_name);
%build_template_structure(template);

%syntax_check;

proc sql noprint;
  select count(*) into :errors from error_messages where error_severity=3;
quit;

%if not &errors %then %do;

%create_title_info(titles_and_footers);
%create_header_info(&denominator_dataset, &by_var, &population);
%create_body_info(&input_dataset);
%create_footer_info(titles_and_footers);

%handle_group_skip_flags();

%split_lines(title);
%split_lines(header);
%split_lines(footer);
%split_lines(body);

%complete_empty_pages;

%construct_report(%if &program_name ne %then %do;%if &output_library ne %then &output_library..;&program_name.&program_suf
%print_report(&output,%if &program_name ne %then %do;%if &output_library ne %then &output_library..;&program_name.&program

%end;

%let end_process=%sysfunc(time());
%let process_time=%sysfunc(putn(%sysevalf(&end_process-&start_process),8.2));

%print_process_overview;

%mend;
```

SHIELD YOUR MACRO FROM BAD INPUT

To prevent user frustration when using your set of macros, you could add checks around parameters you use, to make sure that the input you are getting is actually valid. You could then throw warnings, or even errors, telling the user that his input was invalid.

However, doing this can result in large logs with hidden error messages, forcing the user to skim through the log, often containing several other errors as a result of SAS throwing even more messages into the fray. In this big mess, the user has to try and figure out what actually caused the process to malfunction. These so called black-box macros can cause a huge amount of frustration among the end-users.

A better approach would be to first check all the input given to the process, combining possible errors into a small report, before executing your process. This not only creates a nice overview of what could or would go wrong, it also has some nice perks that come with it:

- Loss of time: your process will not run if its input is not what you expect. Users tend not to expect errors when using macros, so they will never split their datasets into smaller fragments before running your process.

- Locked datasets: Whoever has had this happen to them knows how annoying it is. You work in a SAS environment, create some temporary datasets whose creation may or may not take some time, run a process, only to find out their temporary dataset becomes locked by the macro, and is no longer accessible. Said user often restarts the SAS session, recreates the temporary dataset, and either lives in eternal fear for as long as he uses your macro, or never uses it again.

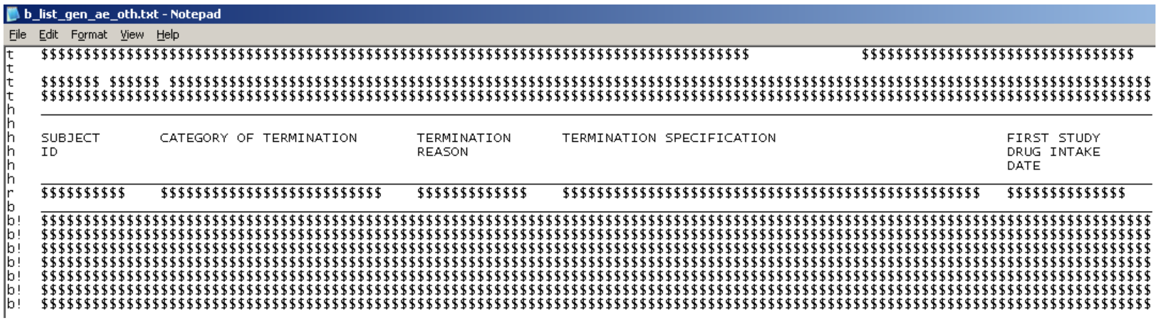
- Missed errors: Missed errors often are far worse than the previous cases. Sometimes users run a process, and after the 10th time, they simply don't bother to check the log anymore. An unexpected situation presents itself, the process does not completely shut down, and the user ends up with a dataset/report that seems good, but really is not. The shield has a separate error report, but only catches what you code it to catch, so caution is always advised.

This shield does not only have to limit itself to process critical errors, but can also provide data feedback. In theory, this means that, while your system may run correctly, even without warnings, you could still provide the user with feedback indicating that their input could hold values that procure results they might not be expecting. An implementing solution for this is to add weight to the errors or warnings you are recording. Allow your process to run when it does not exceed an error threshold, and present the data feedback at the end.

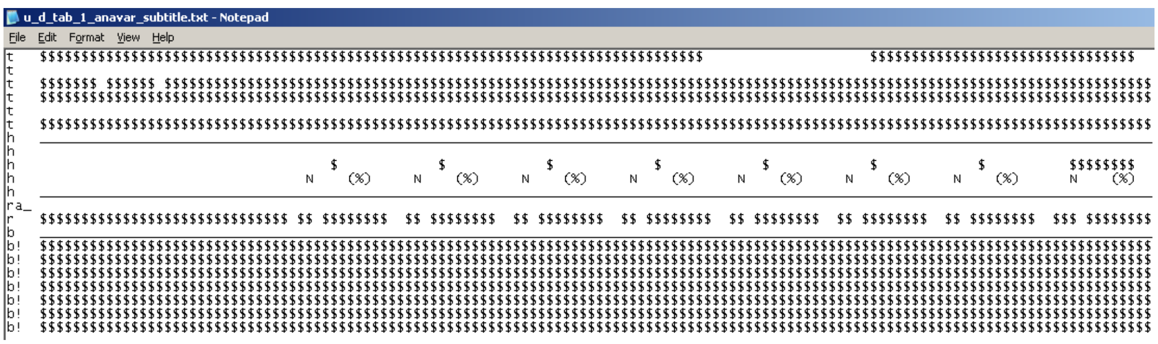
CONFIGURATION

TEMPLATE EXAMPLES

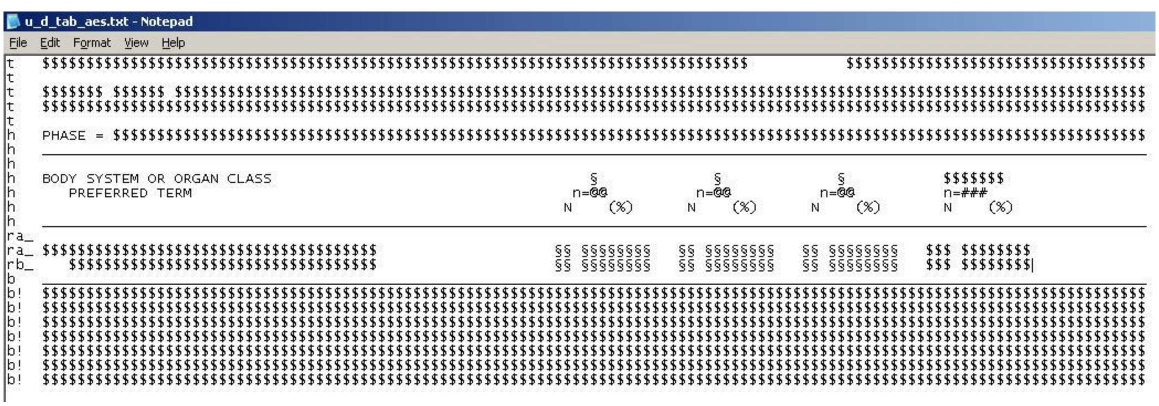
SIMPLE LISTING



SIMPLE TABLE



COMPLEX TABLE



TEMPLATE LAYOUT

- Column 1: flag indicating which subsection of the report the line belongs to.
  This could either be:
  - `t(op)`**:** titles
    - *Input: titles dataset*
  - `h(eader)`**:** display header of columns~,~ like treatment groups and denominator information
    - *Input: denominator dataset*
  - `r(eport)`**:** report data like descriptive statistics, listing, …
    - *Input: input dataset*
  - `b(ottom)`**:** footnotes
    - *Input: titles dataset*

- Column 2-4: control variables.
  - `"!"`**:** If every variable on this line is empty, the line will be deleted.
  - `<x>`**:** This `<x>` is the name of an existing variable in the input dataset. When the value of the variable is set to 1, the line after the control variable will be printed.

- Column 5-…: report layout.
  - A sequence of characters, providing the layout of the report.
  - `'$'` and `'§'` character sequences will be replaced by variables from the input dataset. `'$'` locks the variable to the page, `'§'` is a dynamic variable sequence over several pages.
  - `'#'` character sequences will be replaced with denominator data.
  - `'@date@'` will be replaced by the date at which the report was created.
  - `'@time@'` will be replaced by the time at which the report was created.
  - `'Page x of y'` will enable ~~the~~ page numbering.
  - Other character sequences will be displayed exactly like in the template.

VARIABLE LEVEL CONFIGURATION

Required parameters

- *report_part*

  - *t*: top. Space reserved for titles (automatically populated from the title/footer dataset).
  - *h*: header. Used to print variables in the denominator dataset.
  - *r*: report. Used to print variables in the input dataset.
  - *b*: bottom. Space reserved for footers (automatically populated from the title/footer dataset).

- *variable*

  Describes which variable to use for the ***next*** available variable sequence (for the specified report part), specified in the template.

Optional parameters

- *variable_type*

  Specifies if a variable is a normal variable `(c)` or a denominator value `(n)`.

- *alignment*

  Aligns the values of the variable. Possible alignment values are:
  - `l`: left
  - `r`: right
  - `c`: center
  - `d`: align on dot
  - `n`: display value as in dataset

- *group_skip:* `<x>`

  Where x is the number of the xth grouping variable (see create_report). If a by-variable is present, that value will be #1 and all others will follow. This parameter will not print repeating values for the variable within group `<x>`.

- *split_value*

  Should a value not be able to be printed on the defined space, the process will split it at the last occurrence of the split_value. It can be both single token as character sequence.

- *hard_split*

  Split on the value of the split_value, regardless of the fact that the line fits on the predefined space or not. Possible values are 0 (no) and 1 (yes).

- *default_blank_split*

  Should the split_value not occur in the string, if this parameter is set to 1 (yes), the process will split on the last occurrence of the blank space in the string. Will not execute if set to 0 (no).

CONCLUSION

Due to its design, this system cannot only be easily altered to add functionality, but it can also be subject to quick fixes on a report level. Meeting a customer's needs can simply be a matter of adjusting the template and rerunning the program. This separation between design and layout means that *anyone can use it*. Whereas, in the past, you would have needed a dedicated team of programmers to manually program their reports, you now have a single, or possibly a few end-users, who generate their reports on the fly.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Pascal Hanssens
Business & Decision
St.- Lambertusstraat 141
1200 Brussel

Work Phone: 0478/79.49.08
Email: pascal.hanssens@businessdecision.com

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.