THE GENERATION OF USER DEFINED REPORTS USING SAS® MACROS
Andrew G. Johnstone, Reserve Bank of New Zealand

ABSTRACT
In the past, reports have been generated by mapping on a piece of squared paper exactly how the report is to be laid out. The report is then written using WRITE statements or their equivalents (such as PUT in the SAS system) and specifying line by line the information to be printed, and the columns in which to print them. A user (or programmer) could not actually see the report layout properly until the program was at the testing stage.

Conversely, panel generation routines have advanced to the point where now it is possible to 'paint' a panel so that the panel is generated as the user would see it.

This paper discusses the use of SAS macros to eliminate the report coding step, and the design of a report layout using the 'painting' concept. The views expressed in this paper are entirely those of the author and in no way should be taken to be the official views of the Reserve Bank of New Zealand, the author's employer.

INTRODUCTION
In this paper I will describe the technical history of report generation as I see it, leading into my reasons for attempting to write the SAS macros for a generalised report, and will then, in a step by step fashion, describe how the SAS macros generate the report.

Throughout the paper, the report I will use as an example is a typical telephone account for a New Zealand subscriber. I feel that these macros could be used for any report in a non-standard row/column layout.

HISTORY OF COMPUTER REPORT GENERATION
The First Reports:
These reports were no more than numbers or even codes coming from the machine which then had to interpreted. For example, a primitive report might just be the number 42, which will obviously only have meaning to the person who wrote the program. This number would be the forty second selection of a series of options, the answer to the meaning of life, six times seven, or just about anything at all. In the absence of any meaningful labels the answer is wide open to any interpretation. (It is astounding to think that anyone with hardware and software as sophisticated as the rats in the 'Hitchhiker's Guide to the Galaxy' series could devise a report which had only the answer 'FORTY-TWO'.)

The Advent of Formatted Reports:
With increasingly sophisticated computer languages such as COBOL and FORTRAN, reports with meaningful labels and formats started being produced. To have done this we needed some idea of how the final report was going to be laid out. This was and still is done by the use of line printer spacing sheets. This requires the report to be translated from the spacing sheet to the computer language code. If you have ever worked with these you will know that they are hard to read and to visualise as a completed report. The FORTRAN code which may be associated with this report would be:

```
WRITE(4,10)TODAY
10 FORMAT(1X,'NEW ZEALAND POST OFFICE TELEPHONE/TAX INVOICE
WRITE(4,20)METTLE
20 FORMAT(1X,'REMITTANCE BY POST SHOULD BE
  + ADDRESSED TO P.O. BOX 257 IMPORTANT: THIS
  + ACCOUNT IS DUE AND PAYMENT MUST BE
  + MADE BY: ALLOCATE 257 IN DETAILED
  + FORM FOR THIS PORTION ONLY.' )
```

and this is only for the first few lines. The code shows that a programmer has very little idea of whether the report will turn out as it was intended until the program is completed. This particular type of reporting was closely associated with the idea of a line editor, whereas it was still difficult to see the overall picture even of a program.

Tabular Reports and 4th Generation Languages:
Following the heels of the 3rd generation programming languages such as FORTRAN and COBOL came the 4th generation report writers, which, it is said, makes ad-hoc report writing something for unskilled users. This is true, but the reports available from these languages, or at least the reports that an unskilled user can produce are basic tables of the row/column type. This may not meet exactly the user's requirements (especially as is shown in the case of the telephone account). To produce anything more complex than a row/column table in base SAS requires a more sophisticated knowledge of the SAS system (especially within a DATA step) than just the PROC PRINT statement. The PUT statements within a DATA step can make the code for the report almost, if not exactly, as complicated as the FORTRAN code:

```
data_null:
set report
put 19 'NEW ZEALAND POST OFFICE TELEPHONE/TAX INVOICE
put 257 IMPORTANT: THIS ACCOUNT IS DUE AND PAYMENT MUST BE
  + ADDRESSED TO P.O. BOX 257 IN DETAILED
  + FORM FOR THIS PORTION ONLY.' )
```

Even with this method, some form of line printer spacing chart is needed.

Reporting on information should not be as difficult as and painful a process as that depicted by a cartoon in the January 29, 1986 issue of Punch magazine, where a prisoner is stretched on a rack with the torturer espousing the virtues of his on-line information retrieval system.

Spreadsheets as a Reporting Tool:
With the current trend in computing away from the batch environment to the interactive environment, a great deal of work has been put into screen design for interactive applications. This work has not only been put into the designs of screens that we use in applications such as some PC products and as another example, the SAS session manager, but also into the screen painting tools marketed by some software houses. PROC FSEDIT in SAS/FSP® is such an example. Using this procedure we can modify a screen to edit a SAS data set in the way that we, as the user, want it. In a similar way, modern spreadsheets let you tailor the spreadsheet. These applications are a natural extension of the full screen editor, which is now the most prevalent editor about.

Spreadsheets have some disadvantages to the methods used in things like FSEDIT and, say, ISPF. One is that spreadsheets still exist in the row/column format which means that to display two fields, one below the other, but not aligned:

```
data_null:
set report
put 30 'NEW ZEALAND POST OFFICE TELEPHONE/TAX INVOICE
```

you cannot use the spreadsheet. For this case, you would have to use some other tool, or modify the report to something which is not exactly as you want it. Another disadvantage with the spreadsheet is that if you are only reporting on some data, but not manipulating it (all manipulations already having been performed), then you are virtually using a sledgehammer to crack a walnut.

The Report Painting Concept:
The advantage of being able to 'paint' a screen is all too apparent when using something like FSEDIT, in that you can see what you are getting as you perform the task. The same idea applies to printed reports has the same advantages. But 'painting' also has the advantages over spreadsheets that you can get exactly what you want, and you do not have to learn all the intricacies of a spreadsheet package to produce something as simple as a report.

10.0
23.12
REASONS FOR WRITING THE SAS MACROS

In our organisation, and I suppose in most other organisations, there are a large number of reports which are produced on a periodic basis. I came across one person attempting to automate one particular report using the SAS system. This report did not fit into the standards imposed by PROC PRINT, and so this person attempted to write it using SAS PUT statements within a NULL data set (and incidentally) hoping that although the attempt was successful, the report took a large amount of time to be produced in exactly the correct format. Also, looking at the code which was produced, I felt that surely something could be attempted using the SAS system, it also found when using a well known spreadsheet package, that I had a great deal of difficulty mapping out a report exactly as I wanted it, and sometimes it required what I thought would be a relatively minor change (such as moving some text further across within a row), the problems that I encountered were far greater than expected (I had to redefine column widths, which meant that I then had to shift text in other rows around).

For these reasons I attempted to write something in the SAS system which would overcome these problems.

USING SAS MACROS TO PAINT REPORTS

A report can be separated into two distinct types of fields: the constant field; and the variable field. In the constant fields there is information which is going to be the same regardless of when the report is produced. The variable fields contain changing information and so need some more information about what exactly is to be printed there and in what format.

The method I have used to distinguish between these two types of fields is to define the positioning of a variable field on a report format. I felt that surely something could be attempted using FORTRAN code for other applications with volatile non-tabular report formats. The variables no longer appear in the fields, each is given a name by which they are referred in the report. It is not necessarily the same as the variable which is to be reported in that field. Each name in the report has to be unique unless you want exactly the same information printed in two places. (That is, the same data from the same observation.) This data set is the skeleton data set called TABLE_.

The information then required is given as input to another SAS data set (DEFNS). The information in this data set is:

- the value of a key variable for the substitution;
- the name of the variable defined on the table;
- the name of the variable from the SAS data set to be substituted; and
- whether the output is to be left or right justified in the field.

The data is given in that order on 'cards', with the key value given in a format which the user specifies. This data set I will refer to as the definitions data set.

You will note that the data set to be reported on is supposed to have a key variable within it. The values of this key variable must be unique for each observation. The variable may actually be _n_, the observation number.

Method:
The method used for the system of macros is:

1. Report data set
2. Table skeleton
3. Table definitions
4. Line components
5. Merge definitions and skeleton
6. Merge table and data
7. Create macro variables
8. Output formatted variables
9. Input into VAROUT_
10. Substitute into table
11. Consolidate table line
12. Discard incomplete lines
13. Print table

Steps 1 and 2 occur in the macro called %BODY, step 3 occurs in %DEFNS and the remainder occur in %TABLEEND.

The SAS Macros:

Three parameters are required for %BODY: the name of the SAS data set to be reported on; the DDname of the library in which the skeleton table is held; and the member name of the skeleton. If the member name is not given, the skeleton is assumed to follow the call to the macro and a CARDS statement and its associated semi-colon are required.

Three parameters are required for the %DEFNS macro: the DDname of the library in which the definitions are held; the member name for the definitions; and the informat of the key variable value in the definitions data set. If the member name is not given, the definitions are assumed to follow the call to the macro and a CARDS statement and its associated semi-colon are required.

Only one parameter is required for the %TABLEEND macro and that is the name of the key variable in the report data set.

The Steps in Detail

1. A data set called OUT_, is created as a direct copy of the report data set. This means that the original data is left unchanged.
2. The table skeleton is input. The first column of the skeleton is assumed to be used for carriage control. The skeleton follows the format outlined above.
3. The table definitions are input in the format specified above.
4. A new SAS data set (TABLEOUT_) is created which based on the TABLE_ data set processed in the following manner. Each line of the table is split into 'components'. Each component of the line contains one substitution (apart from the final one) and the point of breakage is where the variable is substituted. As an example take the line:

RENTAL PERIOD:<C ><>D>

This would be broken into 3 components:

RENTAL PERIOD:

(blank line)

Notice at this stage that the variables no longer appear in the components. This is because the positions of the variables are indexed by two new variables in the SAS data set. Another variable (VAROUT_) contains the field name on the table skeleton.

5. The data set DEFNS_ and the data set TABLEOUT_ which now contains the line components are now merged by the variable VAROUT_. This means that each component of the skeleton has the appropriate definition associated with it. This is stored as a new copy of the TABLEOUT_ data set.

6. The data set TABLEOUT_ is merged with the data set OUT_ on the key variable so that each component and associated definition have the appropriate observation associated with them.

7. Macro variables, one for each component (called VARn, where n is the observation number), are created. These contain the name of the variable to be substituted for this component.

8. An external data set (DDname OUTITEMS) is created using the macro variables created in the previous step. This contains, in columns 1 to 4, the observation number of the SAS data set TABLEOUT_ and, in columns 5 onwards the value of the variable to be substituted in this component. If it is the final component on a line, then nothing is output to this data set. The information is output to an external data set to take advantage of the default formats associated with each variable.
9. The information from OUTITEMS is then read back into TABL_OUT which is then merged with TABLOUT by the observation number.

10. The formatted data that was read in from OUTITEMS is then substituted into the appropriate place in the line component using the index calculated previously and the justification variable in the definitions data set.

11. The lines are condensed back into their original form.

12. All but complete lines are discarded.

13. The table is printed to the DName TABLE.

Some points to note about the macros:

- The characters '<' and '>' can be changed to any other characters if required. The only check for these occurs in step 4.

- There is a 40 character maximum length on the substitutable variable. This can be increased or decreased as necessary by changing the format $CHAR40$ to $CHARnn$ where nn is the required length. This occurs in step 9.

- Although I have assumed here that only one report would be made against a data set, which may consist of a number of observations, it is conceivable that you may wish to produce a report (especially in the case of an account) for each observation in a data set. This can be done with only minor changes. Steps 1 to 3 are independent of the data to be tabulated, so that only step 6 onwards need to be repeated.

- The two DNames used (TABLE and OUTITEMS) can be changed to others if necessary. OUTITEMS occurs twice in the code, and TABLE occurs once.

- All datasets and variables (apart from macro variables) used in these macros have an underscore as part of their name.

The SAS Macros

/*---------------------------------------
Table Definition Routine
---------------------------------------*/
* This table definition routine was written by Andrew G. Johnstone Reserve Bank Information Centre. It enables the input to be 'painted' on an output table in exactly the form that the user wishes to see it, and then specify what is to be printed on the form.

The input is in two separate parts, the first being the table as it is to be printed from a single SAS data set which has a key variable within it, which is unique for each observation. This key variable may be a date or it may be _n_ (the observation number) or any other valid variable.

*------------------------------------------------------------------
*** DOBY macro
Purpose: Sets up the dataset to be printed as a temporary dataset, and inputs the table skeleton.
Parameters: DATA - The name of the SAS data set to be reported on.
            INDD - The DDname of the library to read the table skeleton from.
            FILE - The member name of the table skeleton.
            (NOTE: If INDD is not specified, then the input is assumed to be on cards, and the CARDS statement should immediately follow the call to DOBY.)
Method: 1. Creates the data set OUT, as a copy of the data set to be reported on.
2. The table skeleton is input (the first component in the skeleton is a carriage control character). The body of the table is read exactly as it would be printed with the areas for the variable substitutions marked out by '<' and '>'. Each substitute area is given a unique name for the purposes of this table. Each area must not be longer than 40 characters. e.g. A.

*------------------------------------------------------------------
*** DEFNS macro
Parameters: INDD - The DDname of the library to read the table definitions from.
            FILE - The member name of the table definitions.
            (NOTE: If INDD is not specified, then the input is assumed to be on cards, and the CARDS statement should immediately follow the call to DEFNS.)
Method: 1. The table definitions are input. There are 4 variables required to be input for each variables source in the table. The information required by the routine is:
            KEY - Key variable value for this substitution.
            varout. Variable name as defined on the table.
            varin. Variable name in SAS data set to be written.
            justify. Justification L or R for left or right justification.

*------------------------------------------------------------------
*** TABLEOUT(Key) macro
Purpose: Does the processing for the table.
Parameters: KEY - The name of the key variable in the SAS data set.
Method: This will be described as it proceeds.

*------------------------------------------------------------------
**
* This data step breaks the line into components. Each component has one variable substitution on it except for the last component for each line which has none. The components break at the point of substitution. An index of the line where the variables are to be substituted is calculated.

* The variable used in this step are:
  * _lineno_. The line number of the table.
  * _line_. The index of the starting point for the substitution.
  * _index_. The index of the ending point.
  * _lineout_. The line component.

*------------------------------------------------------------------
**
* The next steps are to combine the table body with the table definitions. This is why each line was broken into components in a previous step. The two data sets are merged by the variable varout where it is dropped as it is now no longer necessary.
The data in the table contains the data to be tabulated. This is merged with the data set so that each table line component has the correct data associated with it.

This step creates macro variables, one for each variable in the table. These variables will be used in a loop at a later stage.

This data step creates a temporary external data set with the substituted variables on it in their default print formats. The observation number is also output. The information is then read as character information and condensed back into its original form.

Only the complete lines are kept.

The table is printed.

Author's address:
Andrew G. Johnstone,
Information Centre,
Computer Services Department,
Reserve Bank of New Zealand,
P.O. Box 2498,
Wellington,
NEW ZEALAND.