Three-Step Table Validation

Yang Chen, Forest Research Institute, Jersey City, NJ

ABSTRACT
Table validation could be difficult due to grouping, wrapping and title/footnotes of the actual table. This paper introduces a three-step table validation process as easy as 1-2-3:

1. Use TABLESTR macro to find and read in the Programmer’s table and store table body as a character variable in a dataset.
2. Use DATASTR macro to group, wrap and concatenate validator’s variables corresponding to table columns into one character variable.
3. Use COMPSTR macro to compare the above two datasets.

This process not only eliminates the gruesome manual check and greatly shortens validation time, but also guarantees our validation quality.

INTRODUCTION
In pharmaceutical industry, independent validation is often practiced in order to ensure table quality. The validator produces an output similar to the programmer’s table by following the same table specifications, and compares them.

However, practically table validation could be quite difficult since the programmer’s tables come with titles and footnotes, or the long character strings are usually wrapped, which make it difficult to comparison. Some people choose to do random manual check, which could be very time consuming, and has high chance of missing the real discrepancies when the tables are huge. Some people compare the datasets before outputting to a table, but finding the corresponding variable name could be troublesome and there is no guarantee that the table output will look exactly like the dataset. Some others manually remove the titles and footnotes and split the table output into variables, which require a lot of parameters.

The alternative presented in this paper resolves all the problems above and is composed of only 3 simple steps.

1. Use TABLESTR macro to find the Programmer’s table and convert table body into a dataset.
2. Use DATASTR macro to wrap and concatenate variables corresponding to table columns into another dataset.
3. Use COMPSTR macro to compare the above two datasets and output a comparison result.

STEP 1: FIND AND EXTRACT TABLE BODY
First, we need to locate the target table. Since the programmer can give their tables any name they prefer, TABLESTR macro uses UNIX command to search the programmers’ output directory for any ASCII table files that contains the title. Here below is part of the code:

```%%MACRO TABLESTR(table=, page=1, idsize=0);
... 
DATA _null_; /* Create a list of subdirectories at the user name area */
   CALL SYSTEM("ls -al &rootpath.>t_user");
RUN;
/* read in the user name (col9) excluding directories named as macros, output, program and the validator*/
FILENAME tuser ‘t_user’;
DATA user;
   LENGTH col1-col9 $60;
```
If the table can be found, the table file name, location and creation time will be displayed. If more than one such table exists, the latest one will be used based on table creation time.

Here below is an example:

There are 2 lst files in the user output area for Table 1.1.1.1
However, only the latest produced lst file will be used for comparison

Table 1.1.1.1 looks as:

The macro then extracts the main body of the table. As stated earlier, one of the major obstacles in table validation is to eliminate the grid lines, table title, footnotes, generation time and path. Instead of removing them manually, we achieve this by first locating some key rows in the table. Per our table producing standard, the bottom line of each table page must be "Report generated by program: /sasprog/xyz/xyzmd01/ychen/programs/tables/t14050601.sas Draft 11/29/2006 15:32:00 AM 1 Table 1.1.1.1 Table 1.1.1.1"

Notes:
Report generated by program: /sasprog/xyz/xyzmd01/ychen/programs/tables/t14050601.sas Draft 11/29/2006 15:32:00 AM 1 Table 1.1.1.1 Table 1.1.1.1

Notes:
Report generated by program: /sasprog/xyz/xyzmd01/ychen/programs/tables/t14050601.sas Draft 11/29/2006 15:32:00 AM 1 Table 1.1.1.1 Table 1.1.1.1
Thus irrelevant information such as table title, footnotes, table generation path and time etc. will be removed. Each row of the main body is stored as a string variable in a dataset _tblstr, which looks like:

<table>
<thead>
<tr>
<th>Obs</th>
<th>_tblstr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR Interval (msec)</td>
</tr>
<tr>
<td>2</td>
<td>&gt;= 250 0/16 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/5 (0.0)</td>
</tr>
<tr>
<td>3</td>
<td>QRS Interval (msec)</td>
</tr>
<tr>
<td>4</td>
<td>&gt;= 150 0/16 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/5 (0.0)</td>
</tr>
<tr>
<td>5</td>
<td>QTcB Interval (msec)</td>
</tr>
<tr>
<td>6</td>
<td>&gt;= 500 0/16 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/5 (0.0)</td>
</tr>
<tr>
<td>7</td>
<td>QTcF Interval (msec)</td>
</tr>
<tr>
<td>8</td>
<td>&gt;= 500 0/16 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/5 (0.0)</td>
</tr>
<tr>
<td>9</td>
<td>QTcI Interval (msec)</td>
</tr>
<tr>
<td>10</td>
<td>&gt;= 500 0/16 (0.0) 0/6 (0.0) 1/6 (16.7) 0/6 (0.0) 0/5 (0.0)</td>
</tr>
<tr>
<td>11</td>
<td>PR Interval (msec)</td>
</tr>
<tr>
<td>12</td>
<td>&gt;= 250 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/5 (0.0)</td>
</tr>
<tr>
<td>13</td>
<td>QRS Interval (msec)</td>
</tr>
<tr>
<td>14</td>
<td>&gt;= 150 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0)</td>
</tr>
<tr>
<td>15</td>
<td>QTcB Interval (msec)</td>
</tr>
<tr>
<td>16</td>
<td>&gt;= 500 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0)</td>
</tr>
<tr>
<td>17</td>
<td>QTcF Interval (msec)</td>
</tr>
<tr>
<td>18</td>
<td>&gt;= 500 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0)</td>
</tr>
<tr>
<td>19</td>
<td>QTcI Interval (msec)</td>
</tr>
<tr>
<td>20</td>
<td>&gt;= 500 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 1/46 (2.2) 1/62 (1.6)</td>
</tr>
</tbody>
</table>

However, in this table, since there are 10 columns, they are actually output into two pages. We can interleave them to ‘restore’ it by adding two more options:

%TABLESTR(table=Table 1.1.1.1, page=2, idsiz=30)

In this example, the macro parameter page here means that every two pages need to be merged. In addition, since the header columns of the even pages are identical with the odd ones, they can be removed by cutting the first 30 characters off from the strings of the even pages. The code for this manipulation is omitted here and the merged result is shown below:

<table>
<thead>
<tr>
<th>Obs</th>
<th>_tblstr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR Interval (msec)</td>
</tr>
<tr>
<td>2</td>
<td>&gt;= 250 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0)</td>
</tr>
<tr>
<td>3</td>
<td>QRS Interval (msec)</td>
</tr>
<tr>
<td>4</td>
<td>&gt;= 150 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0)</td>
</tr>
<tr>
<td>5</td>
<td>QTcB Interval (msec)</td>
</tr>
<tr>
<td>6</td>
<td>&gt;= 500 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 0/6 (0.0)</td>
</tr>
<tr>
<td>7</td>
<td>QTcF Interval (msec)</td>
</tr>
<tr>
<td>8</td>
<td>&gt;= 500 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0) 0/6 (0.0)</td>
</tr>
<tr>
<td>9</td>
<td>QTcI Interval (msec)</td>
</tr>
<tr>
<td>10</td>
<td>&gt;= 500 0/6 (0.0) 0/6 (0.0) 0/5 (0.0) 0/6 (0.0) 1/46 (2.2) 1/62 (1.6)</td>
</tr>
</tbody>
</table>
STEP 2: CONCATENATION

Meanwhile, the validator prepares his own dataset through data manipulation according to table specifications. This dataset contains the variables that correspond to the columns of the table. In order to do a comparison, we need to concatenate them into one single string.

In our previous example, suppose the validator prepares a dataset called final with variables cl1 - cl11 corresponding to the columns of PCS Criteria, Placebo, A, B, C, D, H, E, I, G, All non-placebo and all treatments in Table 1.1.1.1.

The concatenation of the above dataset can be easily obtained with one simple data step. When calling the macro DATASTR, simply put a dot between the source dataset and the list of variables to be wrapped.

%DATASTR(final.head cl1 cl2 cl3 cl4 cl5 cl6 cl7 cl8 cl9 cl10 cl11)

It outputs a dataset called _datastr, which contains a character variable called _datastr:

Obs head cl1 cl2 cl3 cl4 cl5 cl6 cl7 cl8 cl9 cl10 cl11

However, it is not rare that the variable is wrapped by PROC REPORT with FLOW option due to line size limit. In addition, if a report contains group variables, with GROUP option, PROC REPORT consolidates into one row all observations from the data set that have a unique combination of formatted values for all group variables. For example, in Table 1.1.1.2, the first several columns are grouped and the Visit Date column is wrapped.

While the validation dataset looks like the datalines below and we would like to wrap the variable visday that corresponds to Visit Date (Day) in the Table 1.1.1.2 and make the repeated information to be displayed as blank so as to imitate the table output.

%DATASTR(final.param/gp pchigh/gp treatc/gp pidc/gp period/gp agesex/gp timerep visit visday/10 valpcs)

The macro DATASTR imitates the FLOW and GROUP options of PROC REPORT by processing variables of the variable list one by one (The running order of each variable is assigned a macro variable i). It creates a separate dataset to store each variable after wrapping and grouping. After all the variables are processed, they are merged and concatenated into a single character variable. The details are show below:

If there is GP option specified, the macro shows only the first record at each group combination level as demonstrated in the code below.
%LET datain=%SCAN(&input, 1, .); /* Gets the source data name */

... DATA &datain;
  SET &datain;
  BY &group; /* List of variables with GP option */
  IF not first.&id&i THEN &&var&i="";
RUN;

If wrapping is requested, the macro will read in the first word of the variable string (if the variable has a format, the macro uses its formatted output) and try to fit it into a temporary string. If there is still space, it will be concatenated with the it, otherwise, the temporary string will be output and reset. This will reiterate until the whole string is wrapped.

The code below does this job:

DATA out&i; /* The wrapped ith variable is output to dataset out&i and they will be merged eventually*/
  SET &datain;
  space1=notspace(&&var&i)-1; /* Counts the numbers of leading spaces of the variable string */
  fstline=1;
  _order=_n_;
RUN;

DATA out&i (KEEP=_order string&i);
  LENGTH string&i _word _word1 $200;
  RETAIN fstline 1; /* Flag whether the current record is the first line or wrapped line.*/
  SET out&i;
  DO WHILE(&&var&i ne "");
    _word=scan(&&var&i, 1, ' '); /* Gets the first word of the current variable string */
    IF fstline ne 1 THEN space2 = space1+1;
    ELSE space2=space1; /* Calculate spaces to be added to each line */
    currspc=notspace(&&var&i)-1; /*Count the current leading spaces in the wrapped lines*/
    IF length(_word)+currspc<length(&&var&i) THEN &&var&i=substr(&&var&i, length(_word)+1+currspc);
    ELSE &&var&i=""; /* Takes the scanned _word out of the string */
    /** If the temporary line is empty, then this is a new line, put the new word into it**/
    IF string&i = " THEN DO;
      IF space2+length(_word)> &width THEN DO;
        _word1=substr(_word, 1, &width-space2);
        &&var&i=strip(substr(_word, &width-space2+1))||&&var&i;
        _word=_word1;
      END; /*Splits long words into two, output the first part and put the rest back to the string */
      string&i = repeat(' ', space2)||strip(_word);
      IF &&var&i="" THEN DO;
        fstline=1;
        OUTPUT;
      END;
    END;
    /**IF the temporary string is not empty and there is still spaces for the new word, concatenate them **/
    ELSE IF length(string&i)+1+length(_word) <=&width THEN DO;
      string&i=trim(string&i)||strip(_word);
      IF &&var&i="" THEN DO; /* If this is the last word of the variable string */
        _order=_order+1;
      fstline = 0;
      OUTPUT;
    END;
  END;
  /** Otherwise output the temporary string and put the _word back into the variable string and reiterate**/
  ELSE DO;
    _order=_order+1;
    fstline = 0;
%COMPSTR

If there is any discrepancy, the differences will be highlighted.

The comparison between Validator and Programmer Output
14:36 Wednesday, December 13, 2006

ORDER SOURCE                  _COMPSTR                  DIFFERENT?
1   Programmer               >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/45(0.0)0/61(0.0) xx
2   Validator                >=1500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/046(0.0)0/62(0.0) xx
3   Programmer               >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/45(0.0)0/61(0.0) xx
4   Validator                >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/045(0.0)0/61(0.0) xx
5   Programmer               >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/45(0.0)0/61(0.0) xx
6   Validator                >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/39(0.0)0/61(0.0) xx
7   Programmer               >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx
8   Validator                >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx
9   Programmer               >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx
10  Validator                >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx
11  Programmer               >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx
12  Validator                >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx
13  Programmer               >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx
14  Validator                >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx
15  Programmer               >=2500/16(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/5(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/6(0.0)0/40(0.0)0/62(0.0) xx

A complete printout of the two datasets is displayed side by side. The source tells whether the string is from Programmer or from Validator. If there is difference, it is flagged as ‘xx’, otherwise ‘OK’ will be flagged.

The wrapped output for Table 1.1.1.2 is below (left aligned) and note that the spaces are removed.

The comparison between Validator and Programmer Output
14:36 Wednesday, December 13, 2006

The difference                     14:36 Wednesday, December 13, 2006   3

A complete printout of the two datasets is displayed side by side. The source tells whether the string is from Programmer or from Validator. If there is difference, it is flagged as ‘xx’, otherwise ‘OK’ will be flagged.

If there is any discrepancy, the differences will be highlighted.

The comparison between Validator and Programmer Output
14:36 Wednesday, December 13, 2006
In our example, due to a manipulation error, the 10th column of the validator is different from the table. After the problem is fixed, the whole table is then considered validated!

<table>
<thead>
<tr>
<th>ORDER</th>
<th>SOURCE</th>
<th>COMPSTR</th>
<th>DIFFERENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Programmer</td>
<td>&gt;=2500/16(0.0)/6(0.0)/5(0.0)/6(0.0)/5(0.0)/6(0.0)/5(0.0)/6(0.0)/45(0.0)/61(0.0)</td>
<td>OK</td>
</tr>
<tr>
<td>1</td>
<td>Validator</td>
<td>&gt;=2500/16(0.0)/6(0.0)/5(0.0)/6(0.0)/5(0.0)/6(0.0)/5(0.0)/6(0.0)/45(0.0)/61(0.0)</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>Programmer</td>
<td>&gt;=1500/16(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/46(0.0)/62(0.0)</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>Validator</td>
<td>&gt;=1500/16(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/46(0.0)/62(0.0)</td>
<td>OK</td>
</tr>
<tr>
<td>3</td>
<td>Programmer</td>
<td>&gt;5000/16(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/46(0.0)/62(0.0)</td>
<td>OK</td>
</tr>
<tr>
<td>3</td>
<td>Validator</td>
<td>&gt;5000/16(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/46(0.0)/62(0.0)</td>
<td>OK</td>
</tr>
<tr>
<td>4</td>
<td>Programmer</td>
<td>&gt;5000/16(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/46(0.0)/62(0.0)</td>
<td>OK</td>
</tr>
<tr>
<td>4</td>
<td>Validator</td>
<td>&gt;5000/16(0.0)/6(0.0)/6(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/46(0.0)/62(0.0)</td>
<td>OK</td>
</tr>
<tr>
<td>5</td>
<td>Programmer</td>
<td>&gt;5000/16(0.0)/6(0.0)/6(0.0)/6(16.7)/6(0.0)/5(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/46(2.2)/62(1.6)</td>
<td>OK</td>
</tr>
<tr>
<td>5</td>
<td>Validator</td>
<td>&gt;5000/16(0.0)/6(0.0)/6(0.0)/6(16.7)/6(0.0)/5(0.0)/6(0.0)/6(0.0)/5(0.0)/6(0.0)/46(2.2)/62(1.6)</td>
<td>OK</td>
</tr>
</tbody>
</table>

CONCLUSION

This set of validation macro utilities: TABLESTR, DATASTR and COMPSTR simplifies the validation process. Although they are run on UNIX platform, this validation approach could be adopted to virtually any operating system. The user just needs to modify the directory/file operating commands accordingly.

Having thee distinct macros allows for greater programming flexibility than one encapsulated macro. Both DATASTR and TABLESTR macro will produce a dataset that the validator can further manipulate to accommodate unexpected circumstances such as reordering, removing or inserting records. In addition, when a validator prefers to output their validation data as a table, macro TABLESTR can be used to read in their table. Whichever way you use these three macros, you will find them easy and efficient in table validation!

CONTACT INFORMATION

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

Yang Chen
Harborside Financial Center Plaza 5
Jersey City, NJ, 07311
Work Phone: (201)427-8294
Fax: (201)427-8496
E-mail: yang.chen@frx.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.
Other brand and product names are trademarks of their respective companies.