PharmaSUG China 2018 – Paper AD-62

Decomposition and Reconstruction of TLF Shells - A Simple, Fast and Accurate Shell Designer

Chengeng Tian, dMed Biopharmaceutical Co., Ltd., Shanghai, China

ABSTRACT

Table/graph shell (mock table/graph) is a blueprint for programmers to generate final Table, Listing and Figure (TLF) of clinical study data. It is usually generated by programmers or statisticians using MS Word or MS Excel and is usually revised multiple times per study team's needs which usually costs a significant amount of time. Our goal is to minimize the time for generating and modifying the shell without the sacrifice of the accuracy and capability of customization. Also, the metadata used in shell generation should be easily reused in the final TLF programming.

The author developed a simple but effective methodology and an application with GUI to quickly generate the metadata for the shell with minimal input needed (stored in XML format for later use) and then call SAS to generate the shell. The whole process generally should be done in less than 10 minutes. As more shell, styles, table parts accumulate, the speed can be even faster. All kinds of TLF shells for SAS can be generated by this application since we use SAS program just like the final TLF. The SAS programs are very flexible that most of the table/listing shells do not require any programming knowledge. The metadata for generating shells are easily modified and reused through the GUI.

INTRODUCTION

In Pharmaceuticals/CROs, generating shell is a necessary process for any reporting work for most of the clinical trials. This paper begins by considering a universal model that all kinds of shells can be generated and managed in a single design. Then we illustrate how a new platform can replace our old generation tools like MS Word. Finally, we introduce some techniques we are using to expedite the process further.

The application was developed in mid-2017 and has been successfully used in dozens of studies from multiple sponsors with thousands of TLF shells created.

CREATING AND STORING THE SHELL METADATA

TABLE SHELL CLASSIFICATION

Let's talk about tables and listings first. For analytical purposes, shell displays are depending on the data type, study design, CRF design and study team's preferences. However, they all share one thing in common. They are all multi-dimensional matrix with group/category names as its dimensions and value (e.g., xx.x for numeric results and HH:MM for the time) as its elements. According to different value patterns, they can be further classified as summary-like tables and listing-like tables.

		Rand Group 1	Rand Group 2
Age	N	x	x
	Mean	xx.x	xx.x
	Min, Max	xx, xx	xx, xx
Sex	Male	xx (xx.x%)	xx (xx.x%)
	Female	xx (xx.x%)	xx (xx.x%)

Shell 1. Summary-like shell. The value pattern for each row of a summary-like shell is unique.

	Age	Weight	Height
10011001	xx	xxx.x	xxx.xx
10011002	xx	xxx.x	xxx.xx

Shell 2. Listing-like shell (listing). The value pattern for each column of a listing-like shell is unique.

		N	Mean	Min, Max
Age	Rand Group 1	x	xx.x	xx, xx
	Rand Group 2	x	xx.x	xx, xx

Shell 3. Listing-like shell (summary)

STORE ANY TLF METADATA WITH THE TREELIKE DATA STRUCTURE

The following table shows a data set example that can be used in generating a shell by SAS, using a normalized data model.

Rand Group 1	Age	N	xx
Rand Group 1	Age	Mean	xx.x
Rand Group 1	Age	Min, Max	xx, xx
Rand Group 1	Sex	Male	xx (xx.x%)
Rand Group 1	Sex	Female	xx (xx.x%)
Rand Group 2	Age	N	xx
Rand Group 2	Age	Mean	xx.x
Rand Group 2	Age	Min, Max	xx, xx
Rand Group 2	Sex	Male	xx (xx.x%)
Rand Group 2	Sex	Female	xx (xx.x%)

Table 1. A simple reporting data set for with a basic data structure

We can generate Shell 1 and Shell 3 using PROC REPORT procedure with this data set, but there are two problems:

- To generate just one data set like Table 1, we need to type repeating information by hand or through copying and pasting. Either way, it costs a significant amount of time.

- We need to write a specific PROC REPORT each time pointing out that which variables are row categories and which variables are column groups.

To deal with these two problems, we introduced a storing/input method that breaks down the data like Shell 1 into smaller parts according to the shell layout which allows the users to quickly build the data needed for the shell and generate the shell using SAS automatically.

The reporting data can be stored as three parts instead of a whole since multi-dimensional data can be created by taking the Cartesian product of sets.

- Row categories sets
- Column groups sets
- Value patterns

Moreover, the row sets and column sets can be presented as a set of many-to-many correspondence combinations as well. This method allows us to create the data set needed for PROC REPORT and generate the correct SAS code at the same time.

The reporting data set *T* can be expressed as:

$$T = A \times B \cdot C$$

where set A is the many-to-many correspondence for all row categories, set B is the many-to-many correspondence for all column groups, set C is the value patterns. $\times$ is the Cartesian product operator while $\cdot$ is the matrix product operator. Please note assuming set A has *m* elements and set B has *n* elements, the C will have $m * n$ elements. (the category element can be either a category or a combination of category and subcategory, likewise for the row group element)

For example, Table 1 can be stored as these three tables: one for all combinations of column groups Table 2, another for all combinations of row categories Table 3, the third for the value patterns Table 4.

Rand Group 1
Rand Group 2

Table 2. Column groups

Age	N
Age	Mean
Age	Min, Max
Sex	Male
Sex	Female

Table 3. Row categories

xx
xx.x
xx, xx
xx (xx.x%)
xx (xx.x%)
xx
xx.x
xx, xx
xx (xx.x%)
xx (xx.x%)

Table 4. Value patterns for each combination of column groups and row categories

Since the value patterns for each row of a summary-like shell is unique, we can further simplify the table by combining the row categories and the value patterns like Table 5:

Age	N	xx
Age	Mean	xx.x
Age	Min, Max	xx, xx
Sex	Male	xx (xx.x%)
Sex	Female	xx (xx.x%)

Table 5. Combination of row categories and value patterns

Likewise, the listing-like value patterns can be stored with column groups table as well.

Please note Table 5 can also be expressed as the Cartesian product of sets (Table 6 and Table 7):

Age

×

N	xx
Mean	xx.x
Min, Max	xx, xx

Table 6. The Age block is expressed as the Cartesian product of 2 sets

Sex

×

Male	xx (xx.x%)
Female	xx (xx.x%)

Table 7. The Sex block is expressed as the Cartesian product of 2 sets

Please notice that Table 1 theoretically can be reconstructed by a combination of Table 2, Table 6 and Table 7 and there will be no redundancy. We can expand this to say that any shell reporting data *T* can be stored as one of the followings:

- Summary-like shell: $T = A \times B_{p\blacksquare}$, where A is all possible combinations of column groups, B_p is all possible combinations of row categories with its corresponding value pattern.
- Summary-like shell: $T = A_p \times B_{\blacksquare}$, where A_p is all possible combinations of column groups with its corresponding value pattern, B is all possible combinations of row categories with its corresponding value pattern.

The rest is easy, we need to find a data structure to store these many-to-many combinations, and it is the tree structure.

e.g.

- Root
    - Age
        - N
            - xx
        - Mean
            - xx.x
        - Min, Max
            - xx, xx
    - Sex
        - Male
            - xx (xx.x%)
        - Female
            - xx (xx.x%)

Furthermore, we have the perfect file format for the tree structure: XML (Extensible Markup Language).

WHAT YOU SEE IS WHAT YOU GET - CREATING THE METADATA WITH A GUI

Since an XML with row categories and column groups information is all we need to create a table shell, we designed a user interface to help create/modify the XML. The interface mainly has three parts:

- Treeview components to display the “row tree” and “column tree”. We can select and check the checkbox on the tree nodes.
- Input components to enter formatted text for one node or a group of nodes
- A group of buttons to do the tree node operations (e.g., add a group of nodes to current level; add a group of nodes to all checked nodes on the tree; remove all checked nodes)

With these components, a user can create the “row tree” and “column tree” from the trunk level to leaf level efficiently. Our application will convert the treeview components to XML format file.

The process can be summarized as below:

1. Add the nodes level by level for both row categories tree and column groups tree.
2. For summary-like shell: add the value patterns to the row categories tree as the lowest level leaf nodes; for listing-like shell: add the value patterns to the column groups tree as the lowest level leaf nodes.
3. Convert the trees on the screen to XML.

TURN METADATA INTO SHELL USING SAS

READING XML WITH SAS

SAS can read XML file and convert it to SAS data set easily using an XML map file. The syntax is like below:

```
FILENAME test 'test.xml';

FILENAME tsmap 'ts.map';

LIBNAME testlib xmlv2 xmlmap=tsmap access=READONLY;

DATA nodes;

    SET testlib.nodes;

RUN;
```

To find more information about creating an XML map file, please refer to this paper.

CREATING THE REPORTING DATA SET BEFORE PROC REPORT

We can use PROC SQL procedure to do a Cartesian product for data sets as follows:

```
PROC SQL;
       CREATE TABLE T AS
       SELECT A.*,B.*
       FROM A CROSS JOIN B;
QUIT;
```

Then we will get a reporting data set with three groups of variables:

- Variables from row categories tree, named as ROWVAR1, ..., ROWVARα
- Variables from column groups tree, named as COLUMNVAR1, ..., COLUMNVARβ
- Value pattern variable named as ROWVARα+1 for summary-like shell; named as COLUMNVARβ+1 for listing-like shell

TURN METADATA INTO RTF AUTOMATICALLY

We can easily create a SAS macro generating the following kind of code for a summary-like shell (α=1, β=2):

```
PROC REPORT DATA=WORK.T SPLIT="|" HEADLINE NOCENTER MISSING ;
   COLUMN  (_page ROWVAR1n ("\ql "  ROWVAR1 ) COLUMNVAR1n,( COLUMNVAR2n ),( ROWVAR2
   dummy ) );
   DEFINE  _page / GROUP FORMAT= BEST9.   NOPRINT RIGHT ORDER=INTERNAL "" ;
   DEFINE  ROWVAR1n / GROUP FORMAT= BEST9.   ID NOPRINT RIGHT ORDER=INTERNAL " " ;
   DEFINE  ROWVAR1 / GROUP FORMAT= $27.   ID LEFT ORDER=INTERNAL " " ;
   DEFINE  COLUMNVAR1n / ACROSS FORMAT= COLUMNVAR1N.   ORDER=INTERNAL " " ;
   DEFINE  COLUMNVAR2n / ACROSS FORMAT= COLUMNVAR2N.   ORDER=INTERNAL " " ;
   DEFINE  ROWVAR2 / DISPLAY FORMAT= $11.   NOZERO " " ;
   DEFINE  dummy / SUM FORMAT= BEST9.   NOPRINT " " ;
RUN;
```

According to different situation, the summary-like shells can be presented as three kinds of layout.

System organ class	preferred term	Group 1(n=xx)	Group 2(n=xx)
SOC1	PT1	xx (xx.x)	xx (xx.x)
	PT2	xx (xx.x)	xx (xx.x)
SOC2	PT1	xx (xx.x)	xx (xx.x)
	PT2	xx (xx.x)	xx (xx.x)

Layout 1. Each level of category takes up a column

SOC PT	Group 1 (n=xx)	Group 2 (n=xx)
Any adverse events, n (%)	xx (xx.x)	xx (xx.x)
SOC1	xx (xx.x)	xx (xx.x)
PT1	xx (xx.x)	xx (xx.x)
PT2	xx (xx.x)	xx (xx.x)
SOC2	xx (xx.x)	xx (xx.x)
PT1	xx (xx.x)	xx (xx.x)
PT2	xx (xx.x)	xx (xx.x)

Layout 2. Just like Layout 1a except there is only one column and different level can be differentiated by indentation.

SOC PT	Group 1(n=xx)	Group 2(n=xx)
System Organ Class a long long name 1		
PT1	xx (xx.x)	xx (xx.x)
PT2	xx (xx.x)	xx (xx.x)
System Organ Class a long long name 2		
PT1	xx (xx.x)	xx (xx.x)
PT2	xx (xx.x)	xx (xx.x)

Layout 3. Only the last level of categories is shown as the first column with proper indentation. The higher level of categories are shown as section title and can be across multiple columns.

Layout 1 and Layout 2 can be generated by a simple PROC REPORT with ACROSS. While Layout 3 can be generated by PROC REPORT with COMPUTE BLOCK.

Meanwhile, listing-like shell only have one layout Layout 4:

	AUClast (ng.hr/mL)	AUCtau (ng.hr/mL)	Cmax (ng/mL)
10011001	xx.xx	xxx.x	xx.x
10011002	xx.xx	xxx.x	xx.x
10011003	xx.xx	xxx.x	xx.x
10011004	xx.xx	xxx.x	xx.x
10011005	xx.xx	xxx.x	xx.x
10011006	xx.xx	xxx.x	xx.x
10011007	xx.xx	xxx.x	xx.x
10011008	xx.xx	xxx.x	xx.x
10011009	xx.xx	xxx.x	xx.x

Layout 4. Value patterns for each column can be different.

Most of the table shells can be categorized as one of the layout mentioned based on author's experience. The user should be able to choose one of the layouts to generate the shell in mind. Calling SAS to generate the RTF is the last step of shell generation.

TECHNIQUES WE ARE USING

- Shell meta parameters settings like titles, footnotes, labels, page type, column headers, column width, pagination, style, fonts are implemented in the interface. And can be imported from external documents like TOC.
- The user can have a real-time preview of the table shell when adding tree node to the "row tree" and "column tree"
- Figure shell is more flexible than the table and listing. However, we managed to develop dozens of macros to generate familiar figures with the similar input as table and listings. If there is new type needed, we need to modify a bit based on the existing macros.
- Some parts of a shell need to be repeatedly added within a table and across tables like descriptive statistics, treatment group labels, codelist in CRF. The application can store these pieces of the shell for later use. The idea is all the information should be reusable and only be entered once.
- Batch search and replace. The shell usually needs to be revised multiple times according to the needs of the study team. Since all the shell meta are based on XML text file, we added the searching and replacing feature.

- Convert XML to RTF shells in batch mode. When all the shell meta files are created/modified/updated, we can use this feature to (re)generate all the shells in a project.
- Control blanks and page breaks. Sometimes part of the table should be set as blank, such as “change from baseline” field for the baseline visit. We use “row category” and “column group” as a 2-dimensional coordinate system to mark any combination of row and column should be set as blank. e.g., if “b”and “3” are marked, the following table will be generated.

	a	b	c	d
1	xx	xx	xx	xx
2	xx	xx	xx	xx
3	xx		xx	xx
4	xx	xx	xx	xx
5	xx	xx	xx	xx

Figure 1. "b"×"3" is marked and set to blanks.

Please note only cartesian product of “row category” and “column group” can be marked. So one set of marks are insufficient. In other words, if the whole table can be expressed as"$abcd$" × "12345", we can only set area like "ad" × "12"(a1, a2, d1, d2) as blank with one set of marks. If we need to set only a1 and d2 as blank, we need two kinds of marks so that the blank area can be expressed as "a" × "1" + "d" × "2". We implement these as different sets of marking buttons.

- Reuse the metadata. The application can load and export table number and title information from Table of Contents; load the codelist/visit information from MetaData Repository (MDR) and autofill when needed; the productive program can read the XML format shell meta for the title/footnotes information to reduce the risk of typo error.

CONCLUSION

The purpose of this application is about saving the time of making final TLF shells and then give us more time for other activities like programming. Spending time up front to develop detailed shells pays off handsomely regarding making final TLF conforms to client needs and then reducing implementation risks. However, drawing tables and graphs using Winword or Excel and typing the data into the computer are not the best use of statistician or programmer’s time. This paper gives us a faster way to create the shells even from scratch.

Some may prefer MS Winword because it feels more intuitive. However, from the experience of the author, the shells for a project seldom finalized in less than two or three versions and the places to be changed are usually a lot. We proved MS Office could be replaced by an all-in-one featured shell designing/managing platform. With the shell meta library accumulating, we may even find a way to convert the SAP to shells directly one day.

REFERENCES

Cisternas, M., Cisternas, R. Reading and Writing XML Files from SAS®, SUGI 29, 2003, http://www2.sas.com/proceedings/sugi29/119-29.pdf

ACKNOWLEDGMENTS

This application was supported by dMed Biopharmaceutical Co., Ltd. I greatly appreciated my colleagues and experts from industry who provided insight and expertise that assisted the development.

CONTACT INFORMATION

E-mail: zzzzqqq@gmail.com