PhUSE 2016

Paper AD01

The TFL Workbench: Tools to Standardise and Accelerate Table, Figure, and Listing (TFL) Programming

Iain Humphreys, PRA Health Sciences, Reading, UK
Hansjörg Frenzel, PRA Health Sciences, Mannheim, Germany

ABSTRACT

In an effort to increase the efficiency, consistency and quality of the table figure and listing (TFL) programming processes, PRA has developed the TFL Workbench tool. Comprising a user front-end for configuring output metadata; the ability to auto-generate reporting code; and a suite of validated SAS® macros, statistical programmers can quickly adapt pre-configured table templates for commonly occurring outputs and automatically generate code to produce the required output.

By describing TFLs in terms of metadata, in configurable output templates, the adoption of TFL standards (both in-house and sponsor) is facilitated, whilst still permitting study-specific customisation. Moreover, the solution is not constrained to any particular data standard or structure, so legacy studies can also be reported.

This paper will share some learning experiences, and describe the solution in terms of opportunities, approach taken, challenges, capabilities, limitations, and future directions.

INTRODUCTION

The traditional approach to programming tables, listings and figures (TFLs) for inclusion in Clinical Study Reports has some widely acknowledged challenges. Inefficiency caused by permitting bespoke programming, rather than re-using code; resource-intensive double-programming for validation; inconsistency of outputs across and within studies; errors caused by misinterpretation of analysis and output specifications; these are all familiar pitfalls associated with the traditional approach.

Many attempts have been made by biostatistical programming departments over the years, with varying degrees of success, to try and automate some or all aspects of TFL production. These have included, amongst others, initiatives such as:

- developing template TFL programs, to be copied and adapted as needed to meet the needs of a specific study;
- developing large SAS macros, intended to replace at least some of the routine programming tasks with calls to pre-written functions.

The goal of achieving TFL automation, however, remains elusive. Common barriers to the wide-scale adoption of solutions exist, and these include: lack of flexibility; complexity of use, including the need to learn a new syntax in some instances; difficulty in validation; resistance amongst some parts of the programmer community to automation.

The establishment and adoption of standards has long been seen as the key to unlocking end-to-end automation in the clinical reporting process, and data standards such as those provided by CDISC are helping to streamline the upstream functions of data capture and subsequent generation of study tabulation and analysis datasets. However, there is yet to emerge any common standard for TFL outputs that would facilitate their automated production.

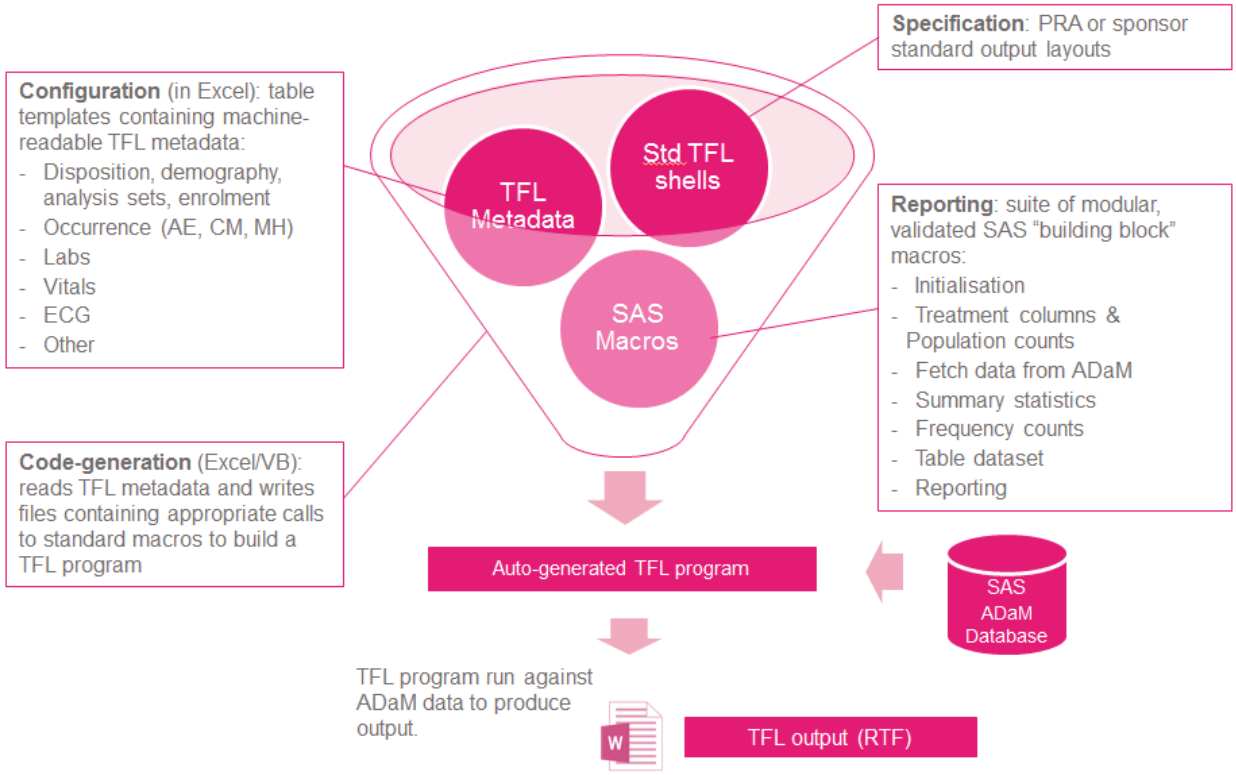
This challenge, significant enough for a study sponsor organization, is magnified further for a Contract Research Organization (CRO), which needs to cater for multiple sponsor standards and reporting requirements. Clearly a new level of flexibility and ease-of-use is needed if efforts at standardizing and automating TFL production are to succeed.

In an attempt to devise a flexible solution that allows standard tables to be defined, and their production automated, PRA has adopted a metadata-driven approach. Taking traditional TFL shells as the specifications for the outputs to be produced, further metadata is collected for each, describing all of the aspects of a TFL output that are typically needed by a biostatistical programmer. This metadata can then be read and processed by a code generator, such that a TFL program is auto-generated, consisting of calls to pre-validated software routines that will produce TFL outputs when they are run.

A review of relevant prior initiatives within PRA was conducted, in order to leverage previous work where possible. This, along with research into other published work [1] in this area, contributed to a set of functional requirements for the various components described above. PRA collectively calls these components the TFL Workbench, and these are described further in the next section.

TFL WORKBENCH COMPONENTS

The TFL Workbench consists of several components working in concert with each other to address many of the issues described above, and to provide a flexible solution for implementing a TFL standard. These components are depicted below.



The **standard set of TFL shells** (PRA or sponsor) are the initial input: these determine which outputs are to be configured in the TFL Workbench to build a 'master' library, and describe their content and their layout. It is especially useful if these shells are annotated in terms of ADaM dataset and variable names, since this greatly facilitates the configuration process.

These specifications guide the **configuring of TFL metadata**, which is carried out in an Excel® workbook. The most commonly occurring and predicable outputs typically reported in clinical studies are described in terms of their metadata, the details of which are discussed further in subsequent sections, with examples. The key point here is that the metadata describes the TFLs in a machine-readable format, which then permits automation. Excel was chosen for its flexibility and ease of use. Once a 'master' library of configurations has been built, the workbook is copied for use on an individual study, allowing the study team to choose from the master set, and permitting a degree of customization to meet study-specific needs. This even extends to creating new configurations, if necessary. This flexibility goes someway to addressing the limitations of some earlier attempts at automation.

Another major component of the TFL Workbench is a **suite of validated macros**, developed in SAS, to perform the various tasks associated with output programming. These are written in separate modules for ease of maintenance, with the main 'building blocks' calling utility macros for lower-level tasks. Error handling is built in, with parameter validation, such that the user is notified of invalid settings, and a clean exit can be made without executing unnecessary subsequent steps. The modular nature also facilitates validation, which is an important feature: it is the

through the deployment of validated software that the need for double-programming is reduced, helping to address another of the drawbacks of current TFL programming practice.

Finally there is the **code generation capability**, which reads the TFL metadata configured in the Excel user interface, and writes TFL programs. These auto-generated programs consist of a sequence of calls to the main 'building block' macros, passing relevant parameters according to the metadata configured. The resulting programs can then be run in SAS to produce TFL output. Individual programs can be produced, e.g. whilst during development; alternatively the facility exists to generate the batch of programs needed to produce all outputs required for an analysis deliverable. A macro 'wrapper' (described in more detail later) ensures repeat tables, i.e. those based on a common configuration, are handled efficiently. The code generator was developed in VBA, and exists within the Excel workbook used to configure the TFL metadata. The rationale for choosing VBA is also discussed in more detail later.

How these components work together is described more fully, with examples, in the following sections.

CONFIGURING OF TFL METADATA

The TFL Workbench is geared towards data standards and output standards; although the TFL Workbench does not rely solely on the ADaM standard to be used, following a data standard facilitates configuring standard templates. Of course, it is still possible to change configurations for individual tables within a study.

The cornerstone of table configuration within the TFL Workbench is the template, as it determines the order and contents of rows in a table. It can be as simple as an adverse events table, but also more complex as a lab shift table or a demography table. Paramount is allowing as much flexibility as possible in the configuration of templates. Other important elements of metadata are population and column definitions. In Excel, the individual templates, and the population and column definitions, are each stored in separate worksheets

Tables are built in the "Table of Outputs" (TOO) worksheet: this contains one row per table. Within this worksheet the user selects the template, the population, the column header set, the dataset to be reported and associated selection criteria, and the treatment variable in the analysis dataset. Additionally, it is possible to specify a row label, for outputs that differ in this regard but which are otherwise repeats, e.g. adverse event tables that differ by the row label of the total number of subjects with events presented in a table.

As an example, the configuration of multiple adverse events tables is described. In the interests of space, a subset of the TFL Workbench columns and rows is displayed below. To illustrate the flexibility of the templates, the configurations for a demography and baseline characteristics table template are also described.

TEMPLATE

Templates make it simpler to set up tables, as they store the core configurations of a table spec. If a table standard is available, templates can already be pre-configured and used by study teams "off the shelf".

ADVERSE EVENTS TEMPLATE

METADATA	SECTION	ROW	DATA TYPE	SUPPRESS IF NONE	ROWLABEL1	INDENT1	TREATMENTS	STATISTIC	PRECISION	CLASS VARS	ANALYSIS VARS	WHERE	DENOM
blank													
row label hdr					System Organ Class # Preferred Term [n (%) m]	0#2							
statistic	1	1	CAT	N	_Row_Label_From_TOO_	0	xxx (xx.x) xxx	n#pct#m	1		COUNT(_ALL_) # COUNT(_ALL_)	TRTEMFL eq 'Y' and AOCCFL eq 'Y' # TRTEMFL eq 'Y'	_BigN_
blank													
statistic	2	1	CAT	N	_CLASS1_	0	xxx (xx.x) xxx	n#pct#m	1	AEBODSYS	COUNT(_ALL_) # COUNT(_ALL_)	TRTEMFL eq 'Y' and AOCCFL eq 'Y' # TRTEMFL eq 'Y'	_BigN_
statistic	2	2	CAT	N	_CLASS2_	2	xxx (xx.x) xxx	n#pct#m	1	AEBODSYS#AEDECOD	COUNT(_ALL_) # COUNT(_ALL_)	TRTEMFL eq 'Y' and AOCCFL eq 'Y' # TRTEMFL eq 'Y'	_BigN_

The sections of a table are entered in column SECTION and its row in column ROW. A section in this context is a (set of) statistics that is generated by one statistical procedure call. The label that describes the row is defined in ROWLABEL1, the indentation of the text in INDENT1. Statistics are defined in columns DATA TYPE, STATISTIC, PRECISION, ANALYSIS VARS, WHERE, and DENOM:

- DATA TYPE – is used to define the type of variable analyzed: CON – continuous variable, CAT – categorical variable
- STATISTICS:
    - For categorical variable: n – counts of subjects, pct – percent values, m – counts of events. Counts are defined together with the entry in column ANALYSIS VARS, CLASS VARS and WHERE: all

records in ADAE are counted (COUNT(_ALL_)), which fulfill the condition in column WHERE and are in the appropriate categories of the classification variables.
    - If there is more than one statistical parameter presented in a row, the keywords are separated by a hash delimiter (#).
    - Percentages are calculated based on the entry in column DENOM; keyword _BigN_ represents the population counts.
- The ANALYSIS VARS column together with the optional value of the WHERE column contain the configuration of what should be analyzed: each row present counts of subjects (n) with treatment emergent AEs (TRTEMFL eq "Y" and AOCCFL eq "Y"), and a count of the events (m) (TRTEMFL eq "Y").
- The CLASS VARS column lists variables that basically act like by-variables, as they limit the counts in each output row to the records that fulfill the combination of values presented in this record.

Column METADATA contains information about the type of row: A record with value "blank" does not provide any function; it is simply entered to improve the readability of the template. "row label hdr" contains the text for the column header of the row label column in the table, and the corresponding indentation for its parts. "statistic" denotes a line containing configurations for statistics. Finally, "label" (not shown in this example) indicates a row in the table which does not present statistics, but might display text, such as a label for a section.

In an occurrence data table usually the categories presented in rows (SOC terms or AE preferred terms in this case) are unknown and the table is largely data-driven. This makes the configuration easy, because the content of the class variables is used to print the row labels: in the case of an adverse events table the variables analyzed are AEBODSYS and AEDECOD. The contents of the row label column is driven by the values of the first and second class variable, depending on the level of data presented (_CLASS1_ to present system organ class names, _CLASS2_ to present preferred terms).

DEMOGRAPHY AND BASELINE CHARACTERISTICS

METADATA	SECTION	ROW	DATA TYPE	SUPPRESS IF NONE	ROWLABEL1	INDENT1	TREATMENTS	STATISTIC	PRECISION	ANALYSIS VARS	WHERE	DENOM
blank												
row label hdr					Characteristic	0						
label	4	1	CON	N	Age (years)	0						
statistic	4	2	CON	N	n	2	xx	n	0	AGE		
statistic	4	3	CON	N	Mean (std)	2	xx.x (xx.xx)	mean#std	0	AGE		
statistic	4	4	CON	N	Median	2	xx.x	median	0	AGE		
statistic	4	5	CON	N	Q1, Q3	2	xx.x, xx.x	q1#q3	0	AGE		
statistic	4	6	CON	N	Min, Max	2	xx, xx	min#max	0	AGE		
blank												
label	9	1	CAT	N	Cigarette smoking (%)	0						
statistic	9	2	CAT	N	Never smoked	2	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'NEVER'	_BigN_
statistic	9	3	CAT	N	Past smoker	2	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'PAST'	_BigN_
statistic	9	4	CAT	N	Current smoker	2	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'CURRENT'	_BigN_
blank												

Within templates, in addition to counts for categorical data, it is also possible to configure descriptive statistics for continuous data. This is indicated by a DATA TYPE of ‘CON’. Labels that indicate the statistical parameter(s) presented are defined in ROWLABEL1, with the indentation of the text in INDENT1. Which statistics are presented in a line and how they are formatted is defined in columns STATISTIC, PRECISION, ANALYSIS VARS, and WHERE:

- STATISTIC: PROC SUMMARY was chosen to produce descriptive statistics, and its statistic keywords can be used to request the results in each line. Again, if multiple results are presented in a row, the keywords are separated by a hash (#).
- PRECISION: In the case of summary statistics this column contains the precision of the values in the input variable. The TFL Workbench increases this value depending on the statistical parameter: e.g. MEAN and MEDIAN are presented with one more digit than the precision value, STD or STDERR are presented with two more digits.
- ANALYSIS VARS: Here the variable that contains the data to produce the statistic is entered. The code generation facility will use the first non-missing entry in this column.
- WHERE: Should there be a reason to filter the observations being summarized, this can be configured in this column.

This example also illustrates the usage of section labels. Rows containing only section labels (e.g. ‘Age (years)’ and ‘Cigarette smoking (%)’) and no statistics can be identified by the value of ‘label’ in column METADATA.

The configuration of the smoking experience results section does not differ substantially from a configuration for adverse events. The major difference is that this section can be used to demonstrate how to configure sections with known outcome categories. Instead of creating the row labels based on the values of a class variable, they are pre-configured in the template, which allows the user to configure rows to be present on the table even if there are no data available for this category, and also to control the order of appearance of categories.

POPULATIONS

Population	Where clause	Data
All Randomized	randfl eq 'Y'	ADSL
Full Analysis Set	fasfl = 'Y'	ADSL
Per Protocol Analysis Set	pprotfl = 'Y'	ADSL
Safety Analysis Set	saffl = 'Y'	ADSL

Populations are defined in the "Populations" tab of the TFL Workbench. The entry consists of a label to identify the individual populations configuration, the name of the subject level dataset used to calculate the count of subjects for treatment columns (population count), and a where clause to filter the records in this dataset appropriately.

When creating a table, these configurations are used to compute the denominators for percent calculation. Because population flags are typically merged to other ADaM datasets, they can also be used to filter the data in other ADaM datasets.

TABLE COLUMNS

Header	Tmt Column	Column Label	Tmt Variable	Value	Strata Variable	Strata Value	Total	Where for _BigN_	No Display
Head_SAF	_c1	Drug A#(N=_BigN_)	TRT01AN	1			N	TRT01AN eq 1	N
Head_SAF	_c2	Placebo#(N=_BigN_)	TRT01AN	2			N	TRT01AN eq 2	N
Head_SAF	_c3	Total#(N=_BigN_)	TRT01AN	1, 2			Y	TRT01AN in(1, 2)	N
Head_SAF_Sex	_c1_1	Drug A#!Female#(N=_BigN_)	TRT01AN	1	SEXN	1	N	TRT01AN = 1 and SEXN = 1	N
Head_SAF_Sex	_c1_2	Drug A#!Male#(N=_BigN_)	TRT01AN	1	SEXN	2	N	TRT01AN = 1 and SEXN = 2	N
Head_SAF_Sex	_c2_1	Placebo#!Female#(N=_BigN_)	TRT01AN	2	SEXN	1	N	TRT01AN = 2 and SEXN = 1	N
Head_SAF_Sex	_c2_2	Placebo#!Male#(N=_BigN_)	TRT01AN	2	SEXN	2	N	TRT01AN = 2 and SEXN = 2	N
Head_Std	_c1	Drug A#(N=_BigN_)	TRT01PN	1			N	TRT01PN eq 1	N
Head_Std	_c2	Placebo#(N=_BigN_)	TRT01PN	2			N	TRT01PN eq 2	N
Head_Std	_c3	Total#(N=_BigN_)	TRT01PN	1, 2			Y	TRT01PN in(1, 2)	N

Table columns other than the label columns are defined in worksheet HEADERS. Each set of columns (mostly treatment columns) is defined by a unique header set name (column HEADER). Each header set can be used for multiple tables. In the output dataset that the TFL Workbench produces, the name of the treatment variables is defined by the values given in columns TMT COLUMN. Values in HEADER and TMT COLUMN uniquely identify each row in the HEADERS tab. The Column label is defined in COLUMN LABEL, the string _BigN_ is a placeholder that is replaced by the appropriate population count. TMT VARIABLE contains the treatment variable to be used to calculate population counts, VALUE contains the value of the treatment variable for each column, and WHERE FOR _BIGN_ the where clause needed to select all rows in the subject level dataset to count subjects in each treatment group.
Stratification can be defined by adding values to columns STRATA VARIABLE and STRATA VALUE, as presented above for header set HEAD_SAF_SEX. Column TOTAL is used to identify total and sub-total columns, NO DISPLAY is set to Y if a results column is to be created, but not presented in the output, which might be the case if the information is used solely for sorting purposes.

CONFIGURING INDIVIDUAL TABLES

In our example, seven repeat adverse events tables are configured. They only use one population definition, one header set definition, and one template definition. The only two configurations that distinguish these tables are the first row labels and the table filters. The TFL Workbench makes it easy to create multiple repeat tables from the same template.

Output ID	Population	Header	Template	Row label	Table Filter	Data Source	Treatment Variable	Treatment as Columns	Stratified (Y/N)	Strata as Columns
14-03-01-02	Safety Analysis Set	Head_SAF	PRA-STD-T-014	Any Adverse Events	TRTEMFL = 'Y'	ADAE	TRT01AN	Y		
14-03-01-06	Safety Analysis Set	Head_SAF	PRA-STD-T-014	Any Treatment-Related Adverse	TRTEMFL = 'Y' and RELGR1N = 1	ADAE	TRT01AN	Y		
14-03-01-10	Safety Analysis Set	Head_SAF	PRA-STD-T-014	Any Adverse Events with an Outcome of Death	TRTEMFL = 'Y' and AESDTH = 'Y'	ADAE	TRT01AN	Y		
14-03-01-12	Safety Analysis Set	Head_SAF	PRA-STD-T-014	Any Serious Adverse Events	TRTEMFL = 'Y' and AESER = 'Y'	ADAE	TRT01AN	Y		
14-03-01-14	Safety Analysis Set	Head_SAF	PRA-STD-T-014	Any Treatment-Related Serious Adverse Events	TRTEMFL = 'Y' and AESER = 'Y' and RELGR1N = 1	ADAE	TRT01AN	Y		
14-03-01-15	Safety Analysis Set	Head_SAF	PRA-STD-T-014	Any Adverse Events Leading to Discontinuation	TRTEMFL = 'Y' and AEACN = 'DRUG WITHDRAWN'	ADAE	TRT01AN	Y		
14-03-01-17	Safety Analysis Set	Head_SAF	PRA-STD-T-014	Any Treatment-Related Adverse Events Leading to Discontinuation	TRTEMFL = 'Y' and AEACN = 'DRUG WITHDRAWN' and RELGR1N = 1	ADAE	TRT01AN	Y		

In the Table of Outputs (TOO) the user configures these tables by specifying the appropriate template, population, and header set, and providing additional information including the Output ID, dataset and treatment variable used, an additional table filter, and the label used for the overall number of subjects and events row.

Different demography and baseline characteristics tables (e.g. presenting data for the ITT set or the Safety Set) can be configured analogously.

Titles and footnotes are currently still managed outside of the TFL Workbench, but their integration is planned for a future phase.

MODIFYING TEMPLATES

Templates can easily be modified by changing the contents of columns, or by adding or deleting rows. The benefit is mostly that a programmer does not need to know how a certain program is constructed to make simple changes. Every programmer trained to use the TFL Workbench learns how to make changes and works in the same framework.

In the example below, the AE template will be updated to only present unique subjects and percentages.

METADATA	SECTION	ROW	ROWLABEL1	INDENT1	TREATMENTS	STATISTIC	PRECISION	CLASS VARS	ANALYSIS VARS	WHERE	DENOM
blank											
row label hdr			System Organ Class # Preferred Term [n (%)]	0#2							
statistic	1	1	_Row_Label_From_TOO_	0	xxx (xx.x)	n#pct	1		COUNT(_ALL_)	TRTEMFL eq 'Y' and AOCCFL eq 'Y'	_BigN_
blank											
statistic	2	1	_CLASS1_	0	xxx (xx.x)	n#pct	1	AEBODSYS	COUNT(_ALL_)	TRTEMFL eq 'Y' and AOCCFL eq 'Y'	_BigN_
statistic	2	2	_CLASS2_	2	xxx (xx.x)	n#pct	1	AEBODSYS#AEDEC(	COUNT(_ALL_)	TRTEMFL eq 'Y' and AOCCFL eq 'Y'	_BigN_

Beside changing the column header (second row) and the TREATMENTS column (which is only for illustration), the only columns that needed updates were the STATISTIC column (change from "n#pct#m" to "n#pct"), the ANALYSIS VARS column (change from "COUNT(ALL)#COUNT(ALL)" to "COUNT(ALL)"), and the WHERE column (deleting the suffix "# TRTEMFL eq 'Y'" from each term).

A further example of template modification is the addition of a sub-total to the display of a categorical variable for baseline characteristics. Doing this in a SAS program usually involves quite some effort. It is a surprisingly easy task within the TFL Workbench. In our Demography and Baseline Characteristics example the smoking experience is presented as:

METADATA	SECTION	ROW	DATA TYPE	SUPPRESS IF NONE	ROWLABEL1	INDENT1	TREATMENTS	STATISTIC	PRECISION	ANALYSIS VARS	WHERE	DENOM
label	9	1	CAT	N	Cigarette smoking (%)	0						
statistic	9	2	CAT	N	Never smoked	2	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'NEVER'	_BigN_
statistic	9	3	CAT	N	Past smoker	2	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'PAST'	_BigN_
statistic	9	4	CAT	N	Current smoker	2	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'CURRENT'	_BigN_

The aim is to add a sub-total of all subject that smoke or have a history of smoking, and base the percentage of past and current smoking on the number of smokers:

METADATA	SECTION	ROW	DATA TYPE	SUPPRESS IF NONE	ROWLABEL1	INDENT1	TREATMENTS	STATISTIC	PRECISION	ANALYSIS VARS	WHERE	DENOM
label	9	1	CAT	N	Cigarette smoking (%)	0						
statistic	9	2	CAT	N	Non-Smoker	2	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'NEVER'	_BigN_
statistic	9	3	CAT	N	Smoker	2	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE in ('PAST' 'CURRENT)	_BigN_
statistic	9	4	CAT	N	Past smoker	4	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'PAST'	ASMOKE in ('PAST' 'CURRENT)
statistic	9	5	CAT	N	Current smoker	4	xx (xx.x)	n#pct	1	COUNT(_ALL_)	ASMOKE eq 'CURRENT'	ASMOKE in ('PAST' 'CURRENT)

For section 9, row 2 the label was changed from "Never smoked" to "Non-Smoker", and as third row a new row was inserted with the label "Smoker". In this row all subjects with smoking experience had to be counted, thus the WHERE column was changed to "ASMOKE in ('PAST' 'CURRENT'). Finally, within the two rows presenting the current smoking habits, the denominator was changed to the number of smokers. Thus categories can be calculated 'on the fly' and do not necessarily need to be present in the analysis dataset being reported.

SUITE OF VALIDATED SAS MACROS

Developing an application like the TFL Workbench with finite resources requires careful consideration is given to which tool is used to achieve a certain task. Writing programs for statistical tables usually requires the following steps:

- Reset the programming environment to a known state
- Create population counts
- Select the data required to produce a table
- Calculate summary statistics
- Combine the results datasets from the previous step into a dataset that has a pre-defined structure, ready to be reported.

- Sort the combined results dataset, add the paging instruction, and print the results to an RTF file.

Because we traditionally use a variety of macros to accomplish these tasks, and because the SAS Macro Language is very capable and allows us to create SAS code depending on the input to macro parameters, we implemented these steps in what we call 'building block' macros. This allows us to keep the SAS TFL programs that will call these macros simple, which is important as we want to create these TFL programs with a code generator. And if the SAS macros can cover a large amount of complexity, the task for the code generator is correspondingly simpler.

In addition to calls to the individual building block macros, TFL Workbench creates a macro 'wrapper' around the definition of a unique table, called %_tflFrame, which is then called at least once; in the case of repeated tables this is called multiple times, to avoid unnecessary code repetition.

SAS CODE GENERATION

The usual choice for the SAS programmer when it comes to generating SAS Code involves the SAS Macro language. This project is no different and in fact relies on validated SAS macros to produce tables. But what is the best choice when generating SAS Macro calls?

Our requirements for the code-generator were the following:

- The configuration data have to be machine-readable. The data must be stored in a format that allows for easy entry/update/storage and also provides a structured way of access when writing SAS programs.
- In addition we need a programming language that is able to read the data, write to text files, and does not have problems with SAS quoting.

Prior experience with projects that used the SAS macro language to generate SAS macro calls has challenged our firm conviction that SAS is the optimal tool for this particular job. While it has the capability to handle most tasks, there are always cases where information assigned to a macro parameter requires quoting, e.g. the expression contains an equal sign or a comma or a macro variable reference that must not be resolved before the macro is called.

We decided to start developing our tool using Excel, for the following reasons:

- It is installed on all our machines, as well as the SAS production environment, which uses Citrix terminal servers and SAS Enterprise Guide employed on a SAS Grid infrastructure
- Excel is familiar to all users, who are to some extend trained to use it
- Excel, as all other Office applications, comes with VBA (Visual Basic® for Applications) build in
- VBA allows – as almost any other programming language – to write text to a file, and thus is able to write SAS programs.
- VBA provides easy access to information stored within Excel

Using Excel and VBA has another surprising benefit. While using SAS would most likely mean that the calls to the building block macros need to be hard-coded in the SAS code that generates SAS TFL programs, this is not at all a necessity when using Excel and VBA. In fact, we treated the macro calls in the same way as other configuration data, and stored all code for macro calls in an Excel worksheet. The VBA code accesses the macro calls the same way it accesses the table metadata, and creates SAS TFL programs by using the metadata that is associated with a particular macro call. Here is an example; the writing of the _tflFrame macro call:

Call ID	SAS Code	Parameter Name	Array Name	Return Type
FRAMESCALL	%_tflFrame(popul = #1	Where clause	POPArray	LIST
FRAMESCALL	, cols = #1	Tmt Column	HeaderArray	LIST
FRAMESCALL	, label = #1	Column Label	HeaderArray	LIST
FRAMESCALL	, nwhere = #1	Where for BigN	HeaderArray	LIST

This allows us to make changes to macro calls without updating the VBA code that writes the SAS program. The TFL Workbench retrieves the SAS code required to write the call to macro _tflFrame from the column SAS Code. It searches the string for an indicator to insert configuration data; in the case of the code above this is the string "#1", which can be found in each line of this call. When writing the macro call to the TFL program file, VBA code replaces the "#1" with the metadata specified in columns "Parameter Name", "Array Name", and "Return Type":

- **Parameter Name:** Name of the column in an Excel worksheet that contains a specific subset of metadata. In the example it is "Where clause" in worksheet "Populations"
- **Array Name:** Name of the location of the metadata, in this case "POPArray" refers to worksheet "Populations".
- **Return Type:** How the metadata are assigned to the macro call. "LIST" returns a pipe-delimited list, if two or more metadata items are returned.

Here is an example: To write the first line of the macro call, the TFL Workbench retrieves the contents of column “Where Clause” from spreadsheet “Populations”.

Population	Where clause	Data
All Randomized	randfl eq 'Y'	ADSL
Full Analysis Set	fasfl = 'Y'	ADSL
Per Protocol Analysis Set	pprotfl = 'Y'	ADSL
Safety Analysis Set	saffl = 'Y'	ADSL

It would assign a pipe delimited list to macro parameter popul=, but the TFL Workbench knows from the TOO worksheet which population is used to create a table, and selects the corresponding where clause by looking up the population in column "Population". As there is only a single value for each population, it simply returns the contents of the appropriate cell of column “Where clause”.

```
%_tflFrame( popul    = %str(saffl = 'Y')
            cols     =  c1| c2| c3
```

As the contents of Where clause for Population “Safety Analysis Set” contains a “=”, the VBA code automatically quotes the expression using macro quoting function %str() and thus declares the “=” as text.

DEFINING THE STRUCTURE OF OUTPUT DATASETS

At PRA all datasets that are used in PROC REPORT statements of summary tables are permanently stored for QC purposes and for possible subsequent use in creating in-text tables. The TFL Workbench creates such datasets by applying a standardized structure:

- SECTION and ROW from the template (_section and _row), which are used (sometimes together with counts or values of class variables to determine the sequence of rows in a PROC REPORT statement.
- Up to two row label columns can be configured (_rtext1 and _rtext2).
- Results are held in columns _c*i* or _c*i*_*j*, where *i* relates to treatment groups, and *j* to categories of a stratification variable. Finally, because they are often used for sorting, the unformatted numeric counts are also added, using the notation _count*i* or _count*i*_*j*, depending on whether results are stratified.

The benefits are clear: tables can easily compared amongst each other, thus it is possible to perform automated consistency checking; tables also can be compared easily between shipments, and differences can be reported in a structured manner, without spending any effort on making the dataset structure unique over all individually programmed tables.

The example below is the input of a table presenting number of subjects in each visit:

	_section	_row	AVISITN	_rtext1	_c1	_c2	_c3	_count1	_count2	_count3
1	1	1	2	Week 1	325 (99.1)	309 (98.4)	634 (98.8)	325	309	634
2	1	1	3	Week 4	302 (92.1)	294 (93.6)	596 (92.8)	302	294	596
3	1	1	4	Week 7	280 (85.4)	276 (87.9)	556 (86.6)	280	276	556
4	1	1	5	Week 10	250 (76.2)	254 (80.9)	504 (78.5)	250	254	504
5	1	1	6	Week 13	243 (74.1)	243 (77.4)	486 (75.7)	243	243	486
6	1	1	7	Week 16	197 (60.1)	209 (66.6)	406 (63.2)	197	209	406
7	1	1	8	End of Treatment^{super a}	325 (99.1)	310 (98.7)	635 (98.9)	325	310	635

In addition the TFL Workbench also stores relevant metadata from the time of table creation, which can be used for many purposes including: to electronically compare the population counts between tables and thus identify errors in the assignment of populations. These are stored in an separate dataset and contain information such as: the name of the source dataset, the name of the dataset on which the population counts are based on, the name of the treatment columns and their related column headers, the number of subjects used as population denominator for each column, and the where clauses used to generate those numbers.

	symbol	value	descr
1	_popwhere	ITTFL eq 'Y'	Population where clause
2	_adam	adam	Libname for source data
3	_poptreat	trt01pn	Population treatment variable
4	_stratvar		Stratifying variable
5	_popdata	ADSL	Dataset containing population set
6	_rtext1hdr	Visit	Column heading for first row label column
7	_n_col	3	Number of column variables
8	_col_1	_c1	Column variable name
9	_col_2	_c2	Column variable name
10	_col_3	_c3	Column variable name
11	_n_1	328	Column population count
12	_n_2	314	Column population count
13	_n_3	642	Column population count
14	_label_1	Drug A#(N=328)#n (%)	Column header
15	_label_2	Placebo#(N=314)#n (%)	Column header
16	_label_3	Total#(N=642)#n (%)	Column header
17	_where_1	TRT01PN eq 1	Population count where clause
18	_where_2	TRT01PN eq 2	Population count where clause
19	_where_3	TRT01PN in (1, 2)	Population count where clause

CONSEQUENCES FOR THE QUALITY: QC OF THE OUTPUT

Traditionally, manually written table programs are subject to a QC process that requires an independent programmer to re-produce the table by writing a QC program. But since the TFL Workbench macros creating the table datasets are validated, this immediately raises the question of what effort is now needed to ensure that the results in the table are those expected when processing the information in the SAP and the table specification? As the macros are validated, we know that they reliably create the same output if the input (ADaM) data remain the same, and if the configurations do not change.

It therefore seems logical to simply review the configurations, since these determine what is produced. With traditionally written SAS programs this is actually quite hard, as the QC reviewer would need to review and understand the SAS syntax in order to determine which dataset is used – for the analyses as well as the population counts, how data are filtered, what procedures are used and the settings applied, and which variables are finally analyzed. With the TFL Workbench the Biostatisticians now have an opportunity to review what has been configured without having to look into individual programs.

It is assumed that having a validated system to generate tables will result in significantly reduced QC effort. This assumption will be tested in further pilot studies.

EXPERIENCE TO DATE

PRA has conducted some pilots on an early version of the TFL Workbench with encouraging results. These pilots were 'retrospective' in the sense that they attempted to recreate the summary statistics for studies already reported. This was carried out to confirm that:

- it was possible to configure the metadata to produce the required tables, in terms of both content and layout;
- the correct summary statistics were created;
- the efforts needed to produce the study tables would be less than using the traditional programming approach.

Target summary tables were those consisting only of descriptive statistics, and not containing any inferential analyses. In both cases a high proportion of target summary tables could be reproduced: the percentage of the non-efficacy tables originally reported in the study could be configured and reproduced by the TFL Workbench ranged from 78% to 98%.

In terms of effort, the configuration time ranged from 3 to 12 hours per unique output, and from 3 to 4.5 hours per table for all outputs (uniques and repeats).

The differences between the results from the pilots can be explained as follows: the outputs generated for the most efficient pilot were closer to the standard set of configurations available at the time, and this pilot had ADaM sets that were considered more 'analysis-ready'. Furthermore the programming resources conducting this pilot were experienced and familiar with the TFL Workbench. In retrospect this pilot therefore represented an optimal case. In contrast, the study that resulted in the least efficient pilot deviated further from the standard configurations at our disposal at the time; had ADaM sets that were less 'analysis-ready', in turn placing a higher burden on the TFL programmer; and was resourced by relatively inexperienced programmers with no prior knowledge of the TFL Workbench. This therefore represents a far less favorable scenario. We expect on average these to represent the most extreme differences for the execution of the TFL creation.

Taken together, the early pilot results suggest significant savings can be made in the production of TFL outputs when certain conditions are met:

- an established standard is configured in the TFL Workbench;
- the study to be reported follows this standard to a reasonably high degree;
- the input datasets are truly analysis ready;
- a little experience is gained in operation of the TFL Workbench.

The feedback from the less experienced programmers was positive, and a preference expressed for controlling TFL generation by configuring metadata, compared with 'manually' programming complex outputs such as shift tables.

FUTURE MEASUREMENTS

Compliance to a given standard will be one of the principal measures, and for this it will be necessary to compare study-level instances of the TFL Workbench with a 'master' version storing the standard. Numeric compliance levels can then be calculated and reported. Understanding what has changed at a study level is key to determining any additional QC steps that may be necessary. It will also assist sponsors in monitoring the uptake of standards, and alert them to any changes that may be required. The TFL Workbench will therefore play an important role in facilitating the introduction and governance of TFL standards.

CONCLUSION

The metadata-driven approach to standardisation and automation of TFLs adopted by the TFL Workbench is showing early promise in overcoming some of the problems and barriers previously seen. Although it is too early to draw solid conclusions backed up by quantitative metrics, initial pilots have delivered favourable results and certainly confirm the concept is feasible. A high proportion of the outputs targeted (those thought to be sufficiently common and predictable in clinical studies) were able to be produced. Ease of use and the flexibility to adjust a standard configuration for study-specific needs were apparent, as was a saving in the time taken to produce summary table outputs. If the barriers to TFL standardisation and automation can be overcome, then not only will cost come down, but quality should increase, in terms of improved consistency and reduced re-work.

REFERENCES

[1] Lyon, Neil. 2015. "The Automated Metadata-driven Table Generation Process (TFLGen) at Amgen Inc." PhUSE 2015, TS08

CONTACT INFORMATION

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

Iain Humphreys
PRA Health Sciences (UK)
500 South Oak Way
Reading, RG2 6AD
Email: HumphreysIain@prahs.com
Web: www.prahs.com

Hansjörg Frenzel
PRA Health Sciences (Germany)
Gottlieb-Daimler-Strasse 10
Mannheim, D-68165
Email: FrenzelHansjörg@prahs.com