PharmaSUG 2019 - Paper AD-203

Large-scale TFL Automation for regulated Pharmaceutical trials using CDISC Analysis Results Metatadata (ARM)

Stuart Malcolm, Frontier Science (Scotland) Ltd.

ABSTRACT

The creation of a Clinical Study Report (CSR) for Phase II/III Pharmaceutical clinical trial involves the production of several hundred Tables, Figures and Listings (TFL). This can be a time-consuming activity when each TFL is programmed manually.

While some TFL are common for many studies, there is always a requirement to create study-specific TFL. In addition, CDISC Analysis Results Metadata (ARM) Define.xml is often requested at the end of the trial.

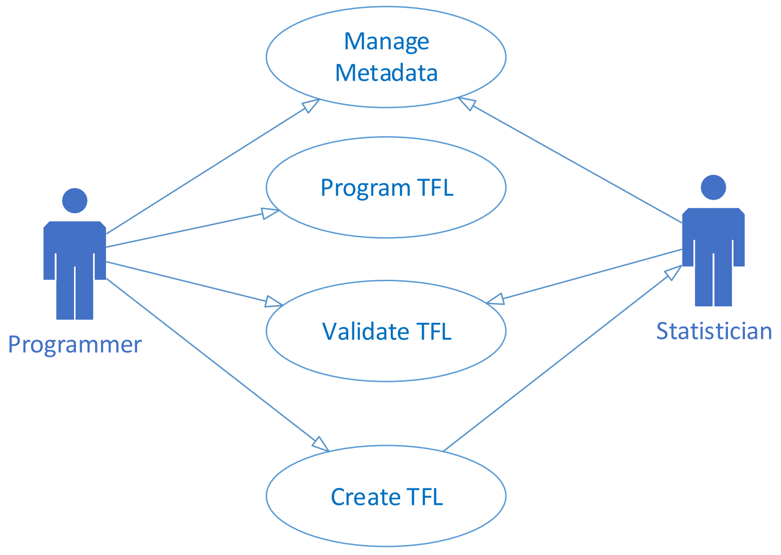
This paper outlines an approach to TFL automation that involved creation of the CDISC Analysis Results Metadata at the start of the process, not the end, and uses this metadata to generate the TFL.

A SAS program structure is described that allows standard TFL to be created while also providing flexibility to easily incorporate study-specific analyses.

INTRODUCTION

The development of TFL is primarily a collaboration between Clinical Statisticians and Statistical Programmers, as shown in Figure 1 below. The Programmer's primary role is to create and QC TFL, and the Statistician's role is to specify the TFL – using a combination of a SAP (Statistical Analysis Plan) and Mock Table Shells.

Figure 1 Users and uses of an automated TFL system



In an automated TFL system, the Metadata performs a similar role as programming specification for traditional manually programmed TFL. Typically the metadata is created by the Programmer and reviewed by the Statistician. The development of the Metadata is the main area of collaboration between Programmers and Statisticians, and one of the benefits of using the CDISC ARM standard for the

Metadata repository is that the ARM Define.xml can be used as a human-readable document during this collaboration.

This paper is intended to provide SAS programmers with an understanding of how to automate the creation of TFL for Clinical Study Reports. Specifically it covers the following key topics:

- The structure of a metadata repository based on the ARM Define.xml standard and extended to support TFL creation (e.g. which TFL are in each Output file, Footnotes, Layout, etc.)
- The use of a 'context' data structure that keeps track of the metadata required to create each TFL of the Clinical Study Report.
- Result Templates (SAS code modules) which use the context to derive the results contained in a TFL. The Report Template applies the 'PROC STATS' to the clinical data (ADaM) to create a Result.
- Creation of output files that contain the Tables, Listings and Figures. PROC DOCUMENT is used to allow TFL be written to multiple output files without having to repeatedly re-run the 'PROC STATS'. This provides significant efficiency savings where there are e.g. several hundred TFL repeated in multiple outputs.

ANATOMY OF A CLINICAL STUDY REPORTING

A Clinical Study Report (CSR) may require the creation of several hundred TFL which are stored in different file formats.

For example, the same eDish plot (i.e. one TFL) may be included in several Output files – in its own RTF file for; in a summary PDF containing all the Safety TFL; and in a PowerPoint slide desk containing top-line results.

The CDISC ARM standard uses the generic term 'Display' for TFL. Each Display contains one or more Result - A Result is e.g. a panel in a table, or the graph in a figure.

Figure 2 Anatomy of a CSR consisting of Outputs, Displays and Results.

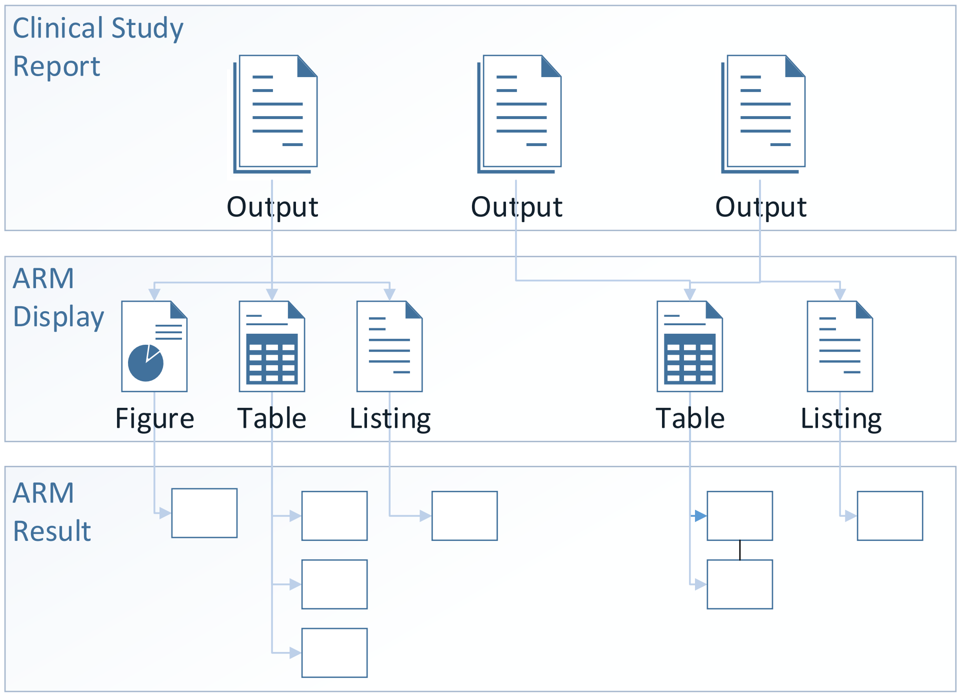


Figure 2 above shows the relationship between Output, Display and Result. This can be viewed as a tree structure with the CSR as the root node, and the Results as leafs. All Output for the CSR can be created by walking this tree structure.

Figure 3 below shows an example Baseline Characteristics table made up of multiple Results. Note that each Display and Result is given a unique identifier that is independent the numbering used in the CSR.

Figure 3 Example TFL annotated with Display and Result identifiers

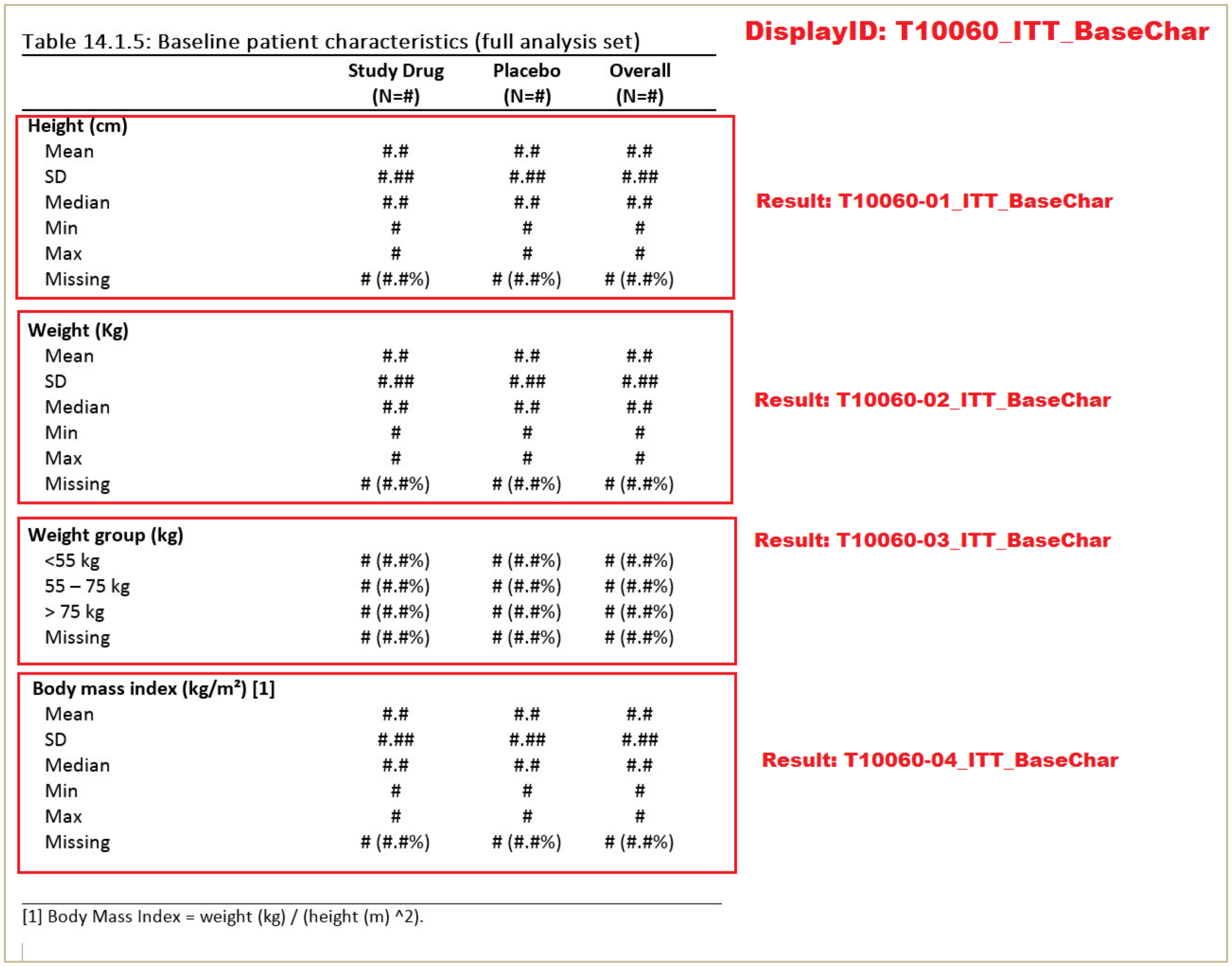


Table 14.1.5: Baseline patient characteristics (full analysis set)

DisplayID: T10060_ITT_BaseChar

	Study Drug (N=#)	Placebo (N=#)	Overall (N=#)
Height (cm)			
Mean	#.#	#.#	#.#
SD	#.##	#.##	#.##
Median	#.#	#.#	#.#
Min	#	#	#
Max	#	#	#
Missing	# (#.#%)	# (#.#%)	# (#.#%)
Weight (Kg)			
Mean	#.#	#.#	#.#
SD	#.##	#.##	#.##
Median	#.#	#.#	#.#
Min	#	#	#
Max	#	#	#
Missing	# (#.#%)	# (#.#%)	# (#.#%)
Weight group (kg)			
<55 kg	# (#.#%)	# (#.#%)	# (#.#%)
55 – 75 kg	# (#.#%)	# (#.#%)	# (#.#%)
> 75 kg	# (#.#%)	# (#.#%)	# (#.#%)
Missing	# (#.#%)	# (#.#%)	# (#.#%)
Body mass index (kg/m²) [1]			
Mean	#.#	#.#	#.#
SD	#.##	#.##	#.##
Median	#.#	#.#	#.#
Min	#	#	#
Max	#	#	#
Missing	# (#.#%)	# (#.#%)	# (#.#%)

Result: T10060-01_ITT_BaseChar

Result: T10060-02_ITT_BaseChar

Result: T10060-03_ITT_BaseChar

Result: T10060-04_ITT_BaseChar

[1] Body Mass Index = weight (kg) / (height (m) ^2).

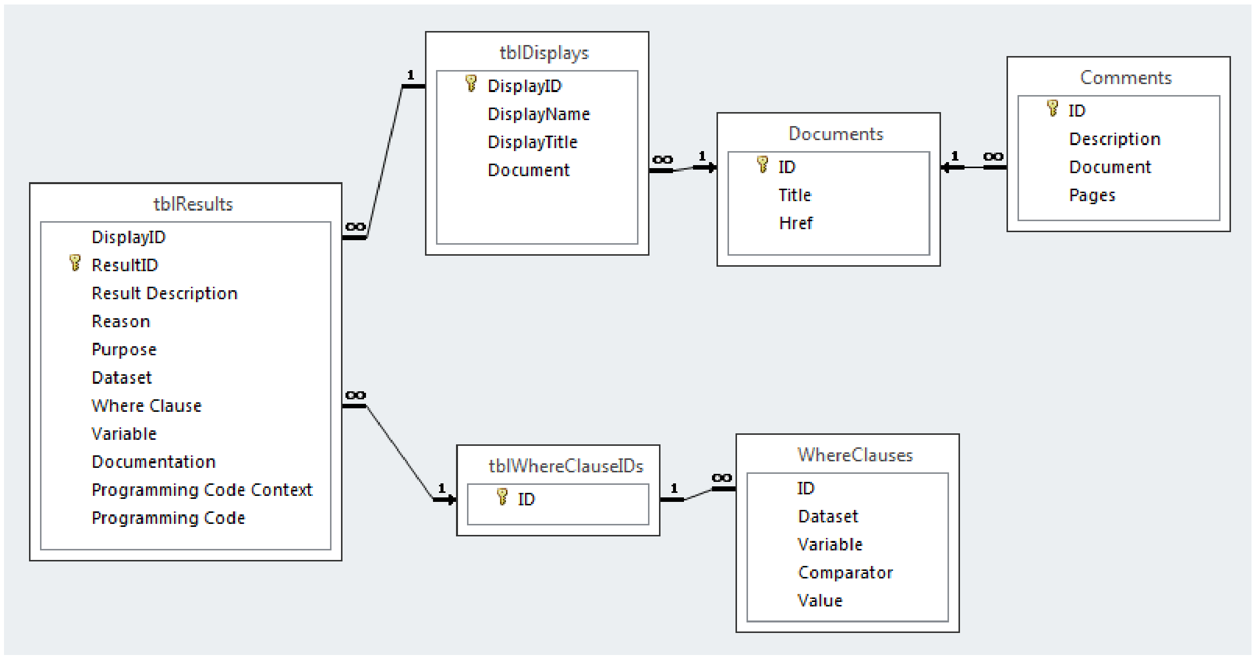
ANALYSIS RESULTS METADATA

All the TFL are created using metadata that is stored in a database (i.e. in the Metadata Repository).

The CDISC ARM Define.xml standard is used as the basis of the Metadata Repository, where the XML structure is 'flattened' into relational tables.

Figure 4 below is the entity relationship diagram for ARM Define.xml tables that have been normalized and created in MS Access.

Figure 4 CDISC ARM is used as basis of the metadata repository database structure



The Metadata Repository becomes the main place where TFL are specified and updated during development. Traditionally this would be done using MS Excel specifications and programming code.

The choice of technology, and the design of a Metadata Repository is outside the scope of this paper, however a project can successfully be piloted using Microsoft Access.

Using CDISC ARM as the metadata schema means that ARM Define.xml can be used for review with Statisticians at an early stage of development, providing better Quality Assurance (QA) and easier validation.

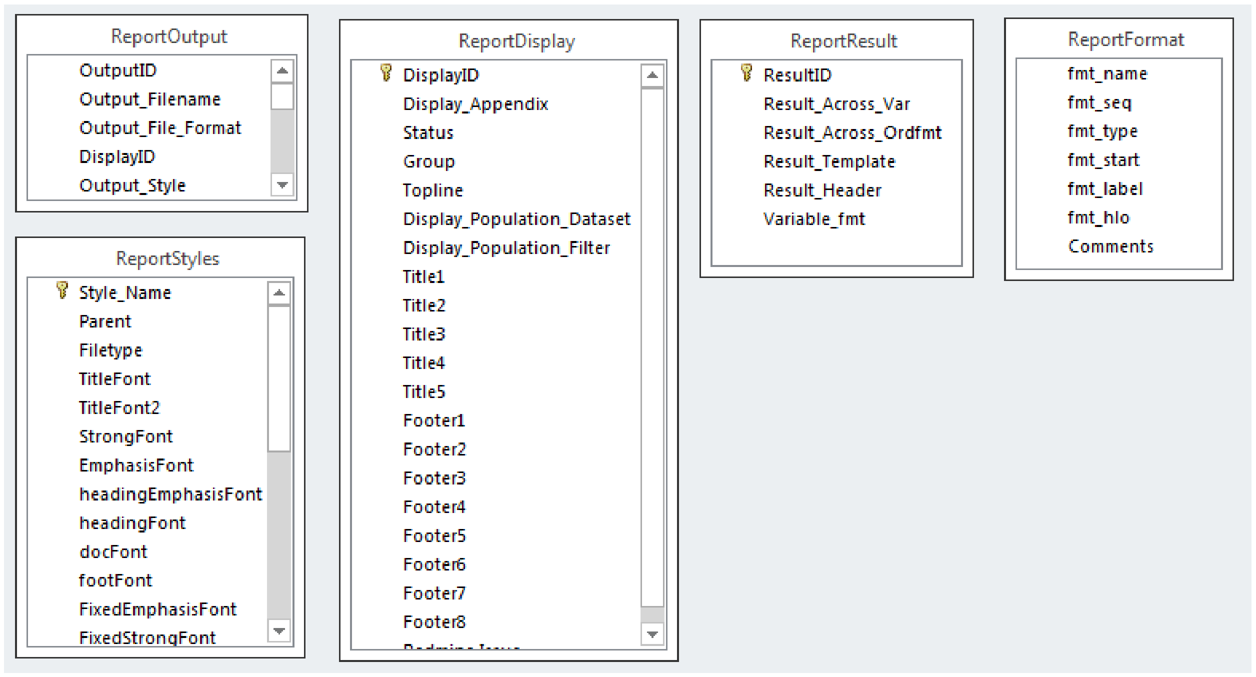
However, ARM metadata is not sufficient to create production-ready TFL. ARM does not contain information on e.g. which TFL go in which Output, Titles, Footnotes, etc.

To support production of TFL, the following additional metadata is required:

- **ReportOutput** – Filenames, file formats, output_style to use for each Output.
- **ReportStyles** – stylesheet configuration for each output_style. This is used to create an ODS style.
- **ReportDisplay** – Headers, footnotes, CSR numbering, how to derive 'big N' for each Display.
- **ReportResults** –How to derive and format the 'across' variable (e.g. treatment arm) and which Result Template to use for each Result.
- **ReportFormat** – Allows SAS Formats to be defined for use by e.g. Across variables, or in Result Templates, etc.

This additional metadata is shown in Figure 5 below.

Figure 5 Additional metadata required for production of Outputs



ReportOutput
- OutputID
- Output_Filename
- Output_File_Format
- DisplayID
- Output_Style

ReportStyles
- Style_Name
- Parent
- Filetype
- TitleFont
- TitleFont2
- StrongFont
- EmphasisFont
- headingEmphasisFont
- headingFont
- docFont
- footFont
- FixedEmphasisFont
- FixedStrongFont

ReportDisplay
- DisplayID
- Display_Appendix
- Status
- Group
- Topline
- Display_Population_Dataset
- Display_Population_Filter
- Title1
- Title2
- Title3
- Title4
- Title5
- Footer1
- Footer2
- Footer3
- Footer4
- Footer5
- Footer6
- Footer7
- Footer8

ReportResult
- ResultID
- Result_Across_Var
- Result_Across_Ordfmt
- Result_Template
- Result_Header
- Variable_fmt

ReportFormat
- fmt_name
- fmt_seq
- fmt_type
- fmt_start
- fmt_label
- fmt_hlo
- Comments

For a more detailed description of all the metadata, see the description of the 'context' data structure later in this paper.

PROGRAM STRUCTURE

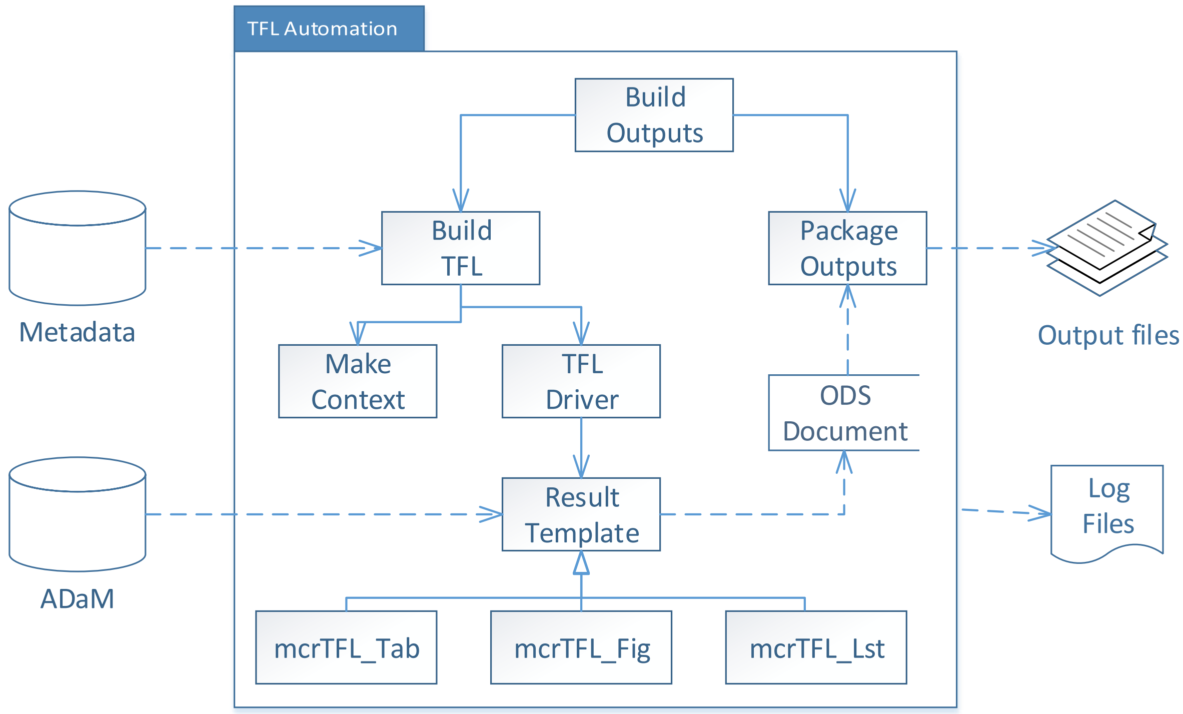
The TFL Automation program structure is based on walking the CSR tree; a context of metadata for each result is accumulated; and when a Result is reached a Result Template macro is called to apply the PROC STATS and create a Result Panel. All panels are 'stacked' for each display, and all displays are written to each output file.

The high-level structure of the TFL Automation programming is shown in Figure 6 below.

The TFL Automation is split into two 'passes' to meet the requirement that although any TFL may be included in multiple Output files we want to derive it only once and simply 'write' it into as many output files as needed.

The two passes are implemented in the 'Build Outputs' module in Figure 6 below, with pass 1 being everything under the 'Build TFL' module and pass two the 'Package Outputs' module.

Figure 6 TFL Automation program structure



During pass 1, all the TFL that are required to populate the Output files are created. PROC DOCUMENT is used to store each TFL in an 'ODS Document' library, and then during the pass 2, the TFL's are 'replayed' out of the ODS Document library and into the requested Output files.

The pseudocode for 'Build Outputs' is:

```
--PASS 1—
   Populate study context
   For each distinct Display in list of Outputs
      Populate display context
      --Build_TFL--
      Create dispatch macro variable
      For each Result in Display
        Populate result context
        Include Driver Program “&ctxDisplay_program”
      End
      Create ODS Document for this TFL
   End

--PASS 2--
   For each Output in list of list of Outputs
      Populate output context
      Load Stylesheet for this Output
      --Package_Output—
      For each Display in Output
         Populate display context
         Replay Display document
      End
      Close Output
  End
```

During Pass 1, a ‘dispatch macro variable’ is created named **`mcrTmpl`**

The value of this variable depends on the type of result being processed. Possible values are:

- `mcrTab_`
- `mcrFig_`
- `mcrLst_`

This is used to implement a ‘dispatch on type’ mechanism, where the TFL Driver programs can call e.g. %&mcrTmpl.open_document and this will actually call %mcrTab_open_document, or %mcrFig_open_document or %mcrLst_open_document depending on whether the current Result is a Table, a Figure or a Listing.

These macros are defined in the Tmpl_Tab.sas, Tmpl_Fig.sas and Tmpl_Lst.sas files (see Table 2 and Table 3 below).

Other than the programming related to scheduling the creation of TFL in two passes, the remainder of the system is concerned with the management of metadata, the application of the appropriate PROC STATS to create the Results, and the creation of the Output files.

These three areas are described in more detail later in this paper.

The modules shown in Figure 6 above are described in Table 1 below:

Table 1 Top-level description of program structure

Program	**Inputs**	**Description**	**Outputs**
Build_Outputs	List of Outputs to create	This is the top-level program. It takes as input a list of one or more output to be created, and implements the ‘two pass’ process described in pseudocode above.	Output files created on disk.
Build_TFL	DisplayID	For any Display, create the TFL by calling each Result Template, and then ‘stacking’ all the Results together to produce an ODS Document that can be replayed by the ‘Package Outputs’ module.	ODS Document
Package_Outputs	List of Outputs to create	Create each output file by ‘replaying’ the ODS Document for each Display contained in that Output; applied output styling, titles, footnotes etc. depending on the Output file format.	Output files created on disk
Make_Context	-	Set of utility macros to read the metadata repository and populate the appropriate context (See CONTEXT section below)	Context
<TFL_Driver>	Context	Each TFL has its own TFL Driver program (named as **<DisplayID>.sas**) Driver programs are created to allow Programmers to customize TFL individually. The macros that each driver program use are defined in one of mcrTFL_Tab, mcrTFL_Fig or mcrTFL_Lst files (see below), and called using the dispatch macro variable (see description above)	ODS Document

		The Driver program is responsible for ‘stacking’ all the Results and creating an ODS Document. This is usually done by calling: **`%&mcrTFL1.stack_results`** An example driver program is given later in this paper.	
<Result_Template>	Context ADaM datasets	Result Templates are SAS macros that are stored in a program name that follows the template naming convention described later in this paper. The Result Template uses the context to apply the appropriate PROC STATS to the ADaM datasets and create a Result that can be stacked. For more detail, see the RESULT TEMPLATES section below.	Interim datasets containing Results
mcrTFL_Tab	-	Defines the macros that are required for Table driver programs. See Table 3 below.	-
mcrTFL _Fig	-	Defines the macros that are required for Figures driver programs. See Table 3 below.	-
mcrTFL_Lst	-	Defines the macros that are required for Listings driver programs. See Table 3 below.	-

Creating a driver program per display means that the Programmer can easily add pre and post processing code to customize individual TFL, or use legacy code, etc.

To avoid duplication of code across driver programs, helper-macros are implemented in mcrTFL_Tab.sas, mcrTFL_Fig.sas and mcrTFL_Lst.sas and a ‘boilerplate’ pattern is used for all driver programs that call these macros using the ‘dispatch macro variable’ called mcrTFL.

The mcrTFL variable is set to either `mcrTab_` `mcrFig_` or `mcrLst_` and is used to call the following macros:

Table 2 Dispatch variable used to call different macros depending on type of TFL

Dispatch value: **Driver macro call**	**mcrTab_**	**mcrFig_**	**mcrLst_**
%&mcrTFL.init_TFL	%mcrTab_init_TFL	%mcrFig_init_TFL	%mcrLst_init_TFL
%&mcrTFL.init_display	%mcrTab_init_display	%mcrFig_init_display	% mcrLst_init_display
%&mcrTFL.derive_result	%mcrTab_derive_result	%mcrFig_derive_result	% mcrLst_derive_result
%&mcrTFL.stack_result	%mcrTab_stack_result	%mcrFig_stack_result	% mcrLst_stack_result
%&mcrTFL.write_display	%mcrTab_write_display	%mcrFig_write_display	% mcrLst_write_display
%&mcrTFL.close_TFL	%mcrTab_close_TFL	%mcrFig_close_TFL	% mcrLst_close_TFL

Using this mechanism means that every driver program can have the same helper-macro calls regardless of the type of Display being created.

The helper-macros all implement the same specification:

Table 3 Table, Figure and Listing helper macros all implement the same specification

Macro	Description
%&mcrTFL.init_TFL	Setup this TFL; setup TFL-specific SAS logging, initialize a new context, and populate context with metadata for study, and display.
%&mcrTFL.init_display	Initialise the ODS Document, setup titles
%&mcrTFL.derive_result	Use the Result Template to create the Result, also create datasets for QC. Generally for figures, this macro will output to ODS Document, however Table and Listings macros will create datasets that will be appended by the stack_results macro and the ODS Document will be written by the write_display macro using PROC REPORT.
%&mcrTFL.stack_result	Primarily for Table (and listings), this macro will append the result datasets, which will be written to ODS Document by the write_display macro.
%&mcrTFL.write_display	Write the display to the ODS Document, write footnotes, etc.
%&mcrTFL.close_TFL	Tidy-up – close the ODS Document, delete the display and result context, remove temporary datasets, etc.

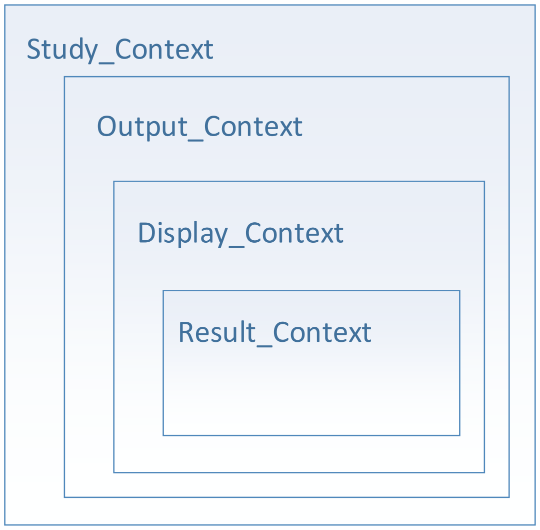
The goal is that all driver programs should be as similar to each other as possible, and only contain SAS code that is necessary to customize that specific TFL. A standard TFL Driver program is shown here:

```
/** BOILERPLATE FOR ALL TFL DRIVER PROGRAMS **/

  * Initialise this TFL;
  %&mcrTFL.init_TFL_program;

  ** ADD DISPLAY PRE-PROCESSING CODE HERE **;

  %&mcrTFL.init_display;

  * for each result in the current display ;
  %do ctxResult_index = 1 %to &ctxDisplay_result_count ;

      * populate the result context ;
     %mcrCTX_populate_result(&&ctxDisplay_result_list&ctxResult_index);

     ** ADD RESULT PRE-PROCESSING CODE HERE **;

     * call the template for this result;
     %&mcrTFL.derive_result;


     ** ADD RESULT POST-PROCESSING CODE HERE **;

     * add the result to the ODS Document ;
     %&mcrTFL.stack_result;

  %end; %* ctxResult_index loop ;

  ** ADD DISPLAY POST-PROCESSING HERE **;

  * write the display, ready for output;
  %&mcrTFL.write_display;

  ** ADD POST-POST PROCESSING HERE **;

  %&mcrTFL.close_TFL;
/** End of program **/
```

IT'S ALL ABOUT CONTEXT

The Context is a data structure that contains all the metadata that is required to create a Result. It contains the metadata from the root of the CSR tree (the Study context) all the way down to the individual Result, as shown in Figure 7 below.

Figure 7 A Context contains all the metadata required to derive a Result



The Context is implemented as SAS macro variables – all named **`ctx<context>_<name>`**

Table 4 SAS Macro variables used to store the metadata Context

Context	**Context Variable**	**Description**
Study	ctxStudy_protocolName	(ARM) Protocol name
	ctxStudy_StandardName	(ARM) CDISC Standard
	ctxStudy_standardVersion	(ARM) CDISC Standard version
	ctxStudy_Name	(ARM) Study name
	ctxStudy_cutoff_date	Data cutoff date (database snapshot/lock)
	ctxStudy_SDTM_version	Version number (from version control system)
	ctxStudy_ADaM_version	Version number (from version control system)
	ctxStudy_TFL_version	Version number (from version control system)
Output	ctxOutput_path	Full path of Output file
	ctxOutput_filename	Output filename (without extension)
	ctxOutput_file_format	Output file format (RTF, PDF, PPTX, etc.)
	ctxOutput_display_list	Space delimited list of DisplayID in this Output
	ctxOutput_display_list1..N	Macro var list with each DisplayID in this Output
	ctxOutput_display_count	Number of Displays in this Output
Display	ctxDisplay_title1..N	Macro var list of title text
	ctxDisplay_footnote1..N	Macro var list of footnote text

	ctxDisplay_population_dataset	Dataset used to derive ‘Big N’ for this Display
	ctxDisplay_population_filter	Filter applied to dataset for ‘Big N’ derivation
	ctxDisplay_dataset	Name of dataset that table results are written to for programmed QC (this is generated by the program and is not read from metadata)
	ctxDisplay_program	Name of the file that contains the TFL Driver program. NOTE that this is not stored in the Metadata but is derived from the ctxDisplay_ID
	ctxDisplay_result_list1..N	List of result ID that are contained in this display. Not defined in metadata – defined in programming.
	ctxDisplay_result_count	Number of Results in this Display. Not defined in metadata – defined in programming.
Result	ctxResult_ID	(ARM) unique ID of result
	ctxResult_Description	(ARM) Description of result
	ctxResult_Reason	(ARM) Rationale for performing the analysis
	ctxResult_Purpose	(ARM) The purpose of the analysis within the body of evidence
	ctxResult_analysis_dataset	(ARM) The source analysis dataset
	ctxResult_where	(ARM) this is read in from the metadata but is re-constructed so that it can be included in a SAS where clause. E.g. `(where=(&result_where))`
	ctxResult_analysis_var	(ARM) variable being analysed
	ctxResult_across_var	E.g. the ‘class’ var in a PROC MEANS. Usually the treatment arm, but can be any analysis variable.
	ctxResult_across_ordfmt	SAS Format name (defined in the ReportFormat metadata) to use to order the across variables. E.g. for treatment arm columns in a table
	ctxResult_template	The template that will create this Result. See the RESULTS TEMPLATES section for more details.
	ctxResult_header	The heading for this Result

NOTE that the `ctxResult_template` variable contains the name of the template to call to generate the Result, it does not contain the results. So, for example, the Result shown in Figure 8 below contains the value “Tmpl_Tab_Con_6stat” – a Table template for a continuous 6 statistic result. Result Templates are described in more detail in the next section.

Figure 8 An example Result annotated with Context variables

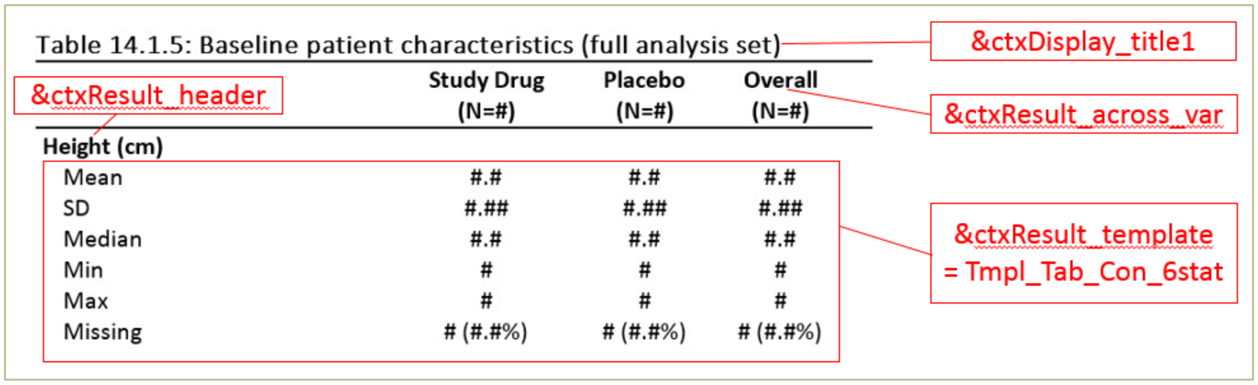


Table 14.1.5: Baseline patient characteristics (full analysis set)

	Study Drug (N=#)	Placebo (N=#)	Overall (N=#)
Height (cm)			
Mean	#.#	#.#	#.#
SD	#.##	#.##	#.##
Median	#.#	#.#	#.#
Min	#	#	#
Max	#	#	#
Missing	# (#.#%)	# (#.#%)	# (#.#%)

&ctxDisplay_title1
&ctxResult_header
&ctxResult_across_var
&ctxResult_template = Tmpl_Tab_Con_6stat

RESULT TEMPLATES

The Result Template is a SAS macro that uses the current Context to apply a 'PROC STATS' to the source data, and creates the Result, it also creates a dataset which is used for QC, as shown in Figure 9 below

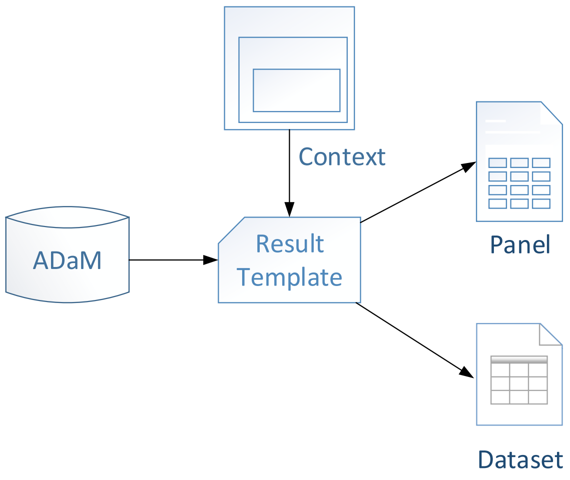


Figure 9 Result Templates are SAS macros that use the Context to create a TFL Panel

Each Result Templates macro is stored in a file with the same name as the template following the naming convention:

```
%Tmpl_<TFL Type>_<View>_<Stat>_<Qualifier>_<nType>
```

Table 5 Result Template naming convention

Template Part	Optional?	Value	Description
TFL Type	No	Tab	Table
		Fig	Figure
		Lst	Listing

View	Yes	Con	Continuous
		Cat	Categorical
		Hier	Hierarchical
		Tab	Tabulate
		Shift	Shift
		Cross	Crosstab
Stat	Depends on View	5stat	Mean, SD, Median, Min, Max
		8stat	N, Mean, SD, Min, Q1, Median, Q3, Max
		2sidedP	P-value calculated from the XXX model
		ciPercent	cumulative incidence
Qualifier	Yes	1d	(default) 1 categorical variable (d=deep)
		2d	2 categorical variables
		3d	3 categorical variables
		1dx1d	1 categorical variable down, 1 cat variable across
		3dx2d	3 categorical variables down, 2 cat variables across

It should be noted that the template to be applied is stored in the metadata, but that the 'PROC STATS' that is used in the template is 'coded' into the template macro, and is not stored in the metadata. Because of this, the 'PROC STATS' is extracted from the template SAS file when the Define.xml is generated by extracting everything between the `/*BEGIN_METHOD*/` and `/*END_METHOD*/` comments.

A Report templates' purpose is to transform source data into a Result. The report template has access to the current Output, Display and Result context - although they will primarily use the current Result context most of the time.

The example code below is indicative of template macro implementation:

```
%macro Tmpl_Tab_Con_5stat;

  %*===========================================;
  %* 5stats continuous result for a Table      ;
  %*===========================================;

  %* use PROC MEANS to derive 5 stats - Mean, Std Dev, Median, Min and Max;

  /* BEGIN_METHOD */
  proc means noprint

    *% setup PROC MEANS using result context variables;
    data=adam.&ctxResult_analysis_dataset
      %if %str(&ctxResult_where) ne %str() %then %do;
        (where=(&ctxResult_where))
      %end; ;

    class &ctxResult_across_var;
    var &ctxResult_analysis_var;

    %* create dataset with raw stats results..;
    %* ..which will be converted into stackable result dataset below;
    output out=raw.& ctxResult_raw_dataset
           mean     = __mean
           stddev   = __sd
           median   = __median
           min      = __min
           max      = __max;
  run;
  /* END_METHOD */

  %*==========================================;
  %* Restructure result_raw_dataset so that   ;
  %* it can be fed into PROC REPORT           ;
  %*==========================================;
  data interim.&ctxResult_standard_dataset;
    set raw.& ctxResult_raw_dataset;

    %* core set of standard result panel variables;
    length RowLabel       $200
           CellTxt Across $20
           AVAL Row       8;

    %* type 0 records are summary of all class vars ;
    if _TYPE_ = 0 then &ctxResult_across_var = "ALL";

    %* output each stat in its own record;
    RowLabel = "Mean";
    AVAL     = __mean;
    Row      = 1;
    CellTxt  = put(aval,best8.1 -C);
    Across   = &result_across_var;
    output;
    %* repeat code above another 4 times..;
    %* ..but with RowLabel = SD, Median, Min and Max.. ;
    %* ..set AVAL = __sd, __median, __min and __max
    %* ..and Row set to 2, 3, 4, 5

    drop __: _FREQ_ &result_across_var;
  run;

  %*==========================================;
  %* Add more processing here as required     ;
  %* for example: set an AcrossOrder var      ;
  %* that can be used to order table columns  ;
  %* ..etc..                                  ;
  %*==========================================;

%MEND;
```

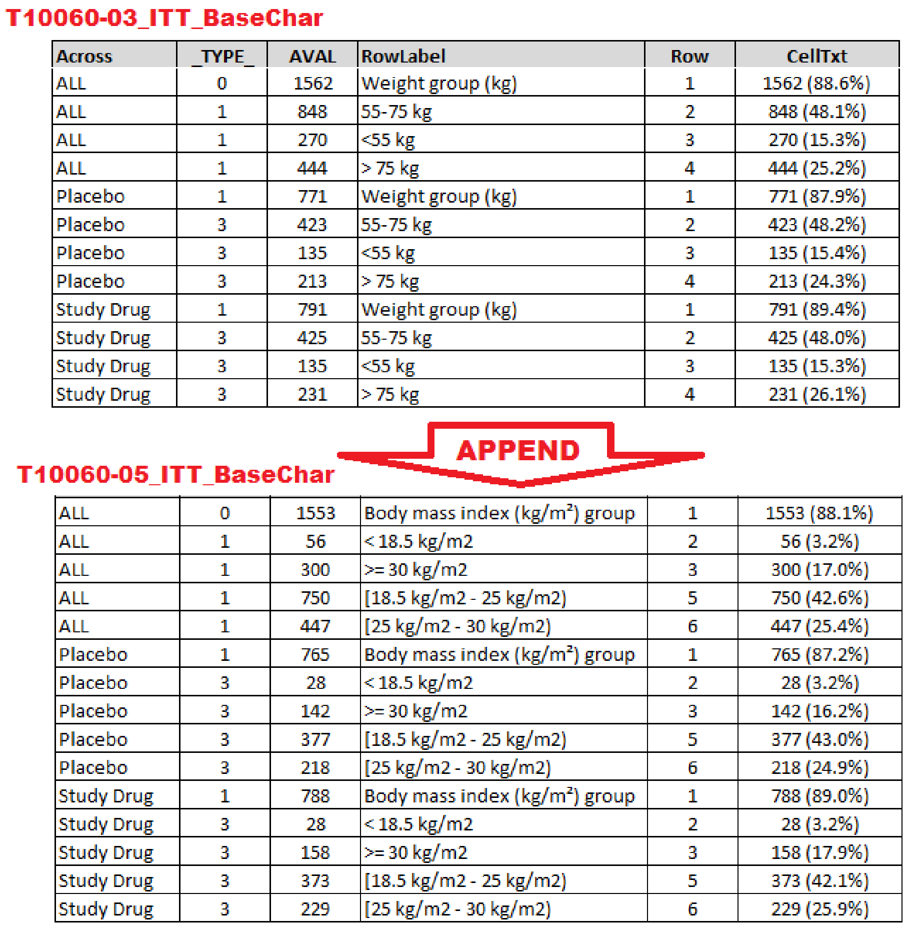
GENERATING OUTPUTS

Result panels that are created by Templates are ‘stacked’ on top of each other to create a Display – this is usually done by the `%&mcrTFL.stack_result` helper macro.

Figures and Listings are relatively straightforward and in effect the Template can create the ODS output directly, however tables is a little more involved, and require the use of an intermediate dataset. These intermediate datasets are then appended together once all the Results in the Table have been created and PROC REPORT is then called from `%&mcrTab_write_display`

An example of how table result panel dataset can be stacked is shown below:

Figure 10 Table results datasets are created that can be appended together



T10060-03_ITT_BaseChar

Across	_TYPE_	AVAL	RowLabel	Row	CellTxt
ALL	0	1562	Weight group (kg)	1	1562 (88.6%)
ALL	1	848	55-75 kg	2	848 (48.1%)
ALL	1	270	<55 kg	3	270 (15.3%)
ALL	1	444	> 75 kg	4	444 (25.2%)
Placebo	1	771	Weight group (kg)	1	771 (87.9%)
Placebo	3	423	55-75 kg	2	423 (48.2%)
Placebo	3	135	<55 kg	3	135 (15.4%)
Placebo	3	213	> 75 kg	4	213 (24.3%)
Study Drug	1	791	Weight group (kg)	1	791 (89.4%)
Study Drug	3	425	55-75 kg	2	425 (48.0%)
Study Drug	3	135	<55 kg	3	135 (15.3%)
Study Drug	3	231	> 75 kg	4	231 (26.1%)

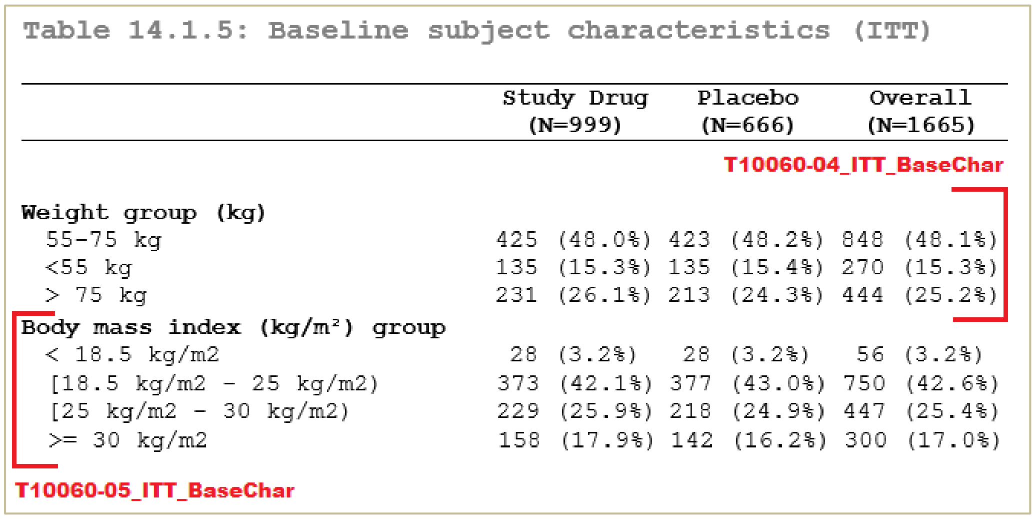
APPEND

T10060-05_ITT_BaseChar

ALL	0	1553	Body mass index (kg/m²) group	1	1553 (88.1%)
ALL	1	56	< 18.5 kg/m2	2	56 (3.2%)
ALL	1	300	>= 30 kg/m2	3	300 (17.0%)
ALL	1	750	[18.5 kg/m2 - 25 kg/m2)	5	750 (42.6%)
ALL	1	447	[25 kg/m2 - 30 kg/m2)	6	447 (25.4%)
Placebo	1	765	Body mass index (kg/m²) group	1	765 (87.2%)
Placebo	3	28	< 18.5 kg/m2	2	28 (3.2%)
Placebo	3	142	>= 30 kg/m2	3	142 (16.2%)
Placebo	3	377	[18.5 kg/m2 - 25 kg/m2)	5	377 (43.0%)
Placebo	3	218	[25 kg/m2 - 30 kg/m2)	6	218 (24.9%)
Study Drug	1	788	Body mass index (kg/m²) group	1	788 (89.0%)
Study Drug	3	28	< 18.5 kg/m2	2	28 (3.2%)
Study Drug	3	158	>= 30 kg/m2	3	158 (17.9%)
Study Drug	3	373	[18.5 kg/m2 - 25 kg/m2)	5	373 (42.1%)
Study Drug	3	229	[25 kg/m2 - 30 kg/m2)	6	229 (25.9%)

Once all the result panel datasets are stacked, PROC REPORT is used to write the table to the ODS Document. For example, the results shown above could create the table fragment shown below:

Figure 11 PROC REPORT is used to create tables from stacked results datasets



```
Table 14.1.5: Baseline subject characteristics (ITT)

                                    Study Drug    Placebo      Overall
                                     (N=999)      (N=666)      (N=1665)
                                                        T10060-04_ITT_BaseChar

Weight group (kg)
  55-75 kg                         425 (48.0%) 423 (48.2%) 848 (48.1%)
  <55 kg                           135 (15.3%) 135 (15.4%) 270 (15.3%)
  > 75 kg                          231 (26.1%) 213 (24.3%) 444 (25.2%)
Body mass index (kg/m²) group
  < 18.5 kg/m2                      28 (3.2%)   28 (3.2%)   56 (3.2%)
  [18.5 kg/m2 - 25 kg/m2)          373 (42.1%) 377 (43.0%) 750 (42.6%)
  [25 kg/m2 - 30 kg/m2)            229 (25.9%) 218 (24.9%) 447 (25.4%)
  >= 30 kg/m2                      158 (17.9%) 142 (16.2%) 300 (17.0%)

T10060-05_ITT_BaseChar
```

CONCLUSION

This paper provides an overview of a metadata-driven approach to the creation of TFL for Phase II/III Pharmaceutical clinical trial. The main benefits over the traditional approach of manually programming each TFL are:

- **Standards Based** – using CDISC ARM Define.xml as the structure for the metadata repository provides a future-proof TFL implementation, and contributes to an 'end-to-end standards' strategy
- **Efficiency** - no matter how many TFL there are, and how many TFL appear in multiple output files, each TFL is derived once, and outputs can be re-generated individually and as a batch without duplicate processing.
- **Increase quality** - programming can be written and implemented independently from the TFL and pre-validated for use across studies.
- **Robustness** - programming can include 'metadata runtime checking' to detect and catch mis-specification in the metadata repository or data quality issues.
- **Better validation** - the CSR outputs can be reviewed by Statisticians using ARM Define.xml documentation rather than having to walk review Excel specifications or SAS code.
- **Reduced development time** - new TFL can be created using Result Templates and Driver program boilerplate more quickly than 'starting from scratch'.
- **New possibilities** –Metadata-driven TFL creation opens new possibilities such as the creation of interactive/hypermedia CSR reports, improved metrics and benchmarking, TFL creation tools, etc!

RECOMMENDED READING

- Analysis Results Metadata (ARM) v1.0 for Define-XML v2.0. www.cdisc.org
- Peter van Reusel and Sam Hume. CDISC Proof of Concept: Evolving our standards towards end to end automation. www.cdisc.org.
- Humphreys, Iain and Frenzel, Hansjörg. PhUSE 2016. "The TFL Workbench: Tools to Standardise and Accelerate Table, Figure, and Listing (TFL) Programming". Paper AD01
- Frank Dilorio and Jeffrey Abolafia. PharmaSUG 2016. "Results-Level Metadata: What, How, and Why" Paper DS03
- Carla Santillan. PhUSE 2016. "Analysis Result Metadata… are we there yet?" Paper CD11
- Mark Crangle. PhUSE EU Connect 2018. "Generating Define.xml and Analysis Result Metadata using Specifications, Datasets and TFL Annotation". Paper DS07

CONTACT INFORMATION

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

Stuart Malcolm
Frontier Science (Scotland) Ltd.
Stuart.Malcolm@frontier-science.co.uk
www.frontier-science.co.uk
www.linkedin.com/in/stuart-malcolm/