PharmaSUG China 2022 - Paper AD-134

Interlocking and Mutually Reinforcing System - From Shells to TLGs Automatically Generation

Zhiping Yan, Dizal Pharma

ABSTRACT

Drawing TLG (Table, Listing and Graph) mock shell relies heavily on manual work. After spending a lot of time on generation, modification, and finalization the mock shell, statistical programmers still need to manually write repetitive code for creation of TLG.

To reduce manual work and improve efficiency, we designed a metadata driven process to automate generation of mock shell as well as TLG. By adopting Python GUI (Graphical User Interface) toolkit and NLP (Natural Language Processing), this process was implemented with two applications. With these applications, statistical programmers can create shell and final output within short time by just clicking on several buttons.

INTRODUCTION

Scalability and reusability of metadata-driven solution used in statistical programming can help avoid inconsistencies and issues with data analysis and reporting across studies as well as reduce redundancy and unnecessary use of space. More importantly, having the metadata centralized can make programming task less time-consuming.

The working efficiency of statistical programmers can be further improved by leveraging application — Graphical User Interface (GUI) — developed by Python. The operations — like clicking on buttons, selecting combo box item — over applications can seamlessly connect the metadata system and whole TLG automation process.

TLG AUTOMATION PROCESS

With TOC provided by statistician and global metadata files derived from global standard shell, study level XML metadata and excel file containing title and footnote can be generated. These two kinds of material can be used to generate study level mock shell. With help of NLP (Nature Language Processing) like NER (Name Entity Recognition), key words and phrases can be extracted from TLG title and mock shell. The extracted key words/phrases and SAS macro mapping metadata can be linked together with fuzzy match or machine learning to generate TLG macro parameter values. The generated macro parameter values can be filled into SAS program template and thus create study level TLG SAS program file which can generate final TLG.

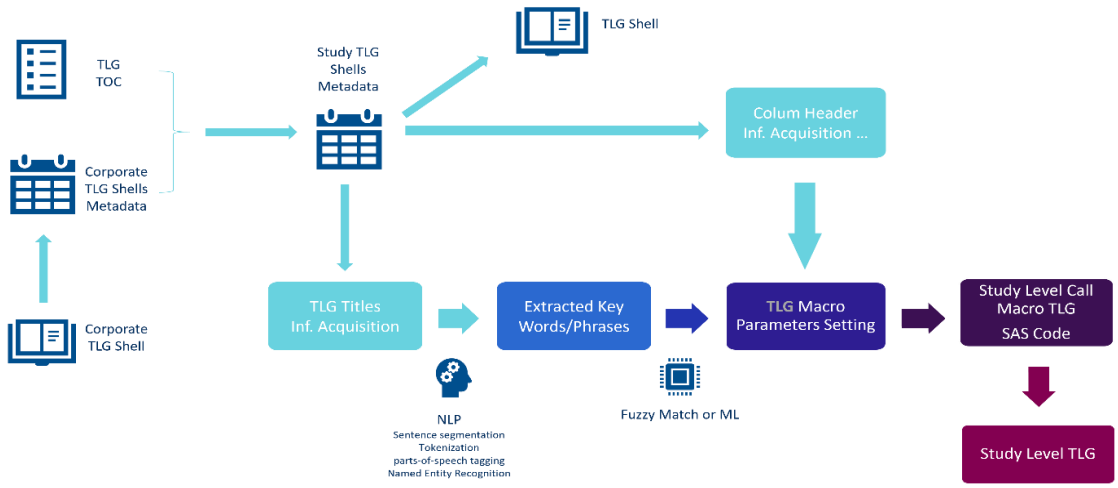


Figure 1. TLG Automation Process

TLG APPLICATION DEVELOPMENT

Above TLG automation process and post TLG processing can be linked by four applications, Generate Mock Shell, CDARS, Combine TLG and CSR Intext Table Insertion. Generation, usage, recycling of metadata is also managed quietly behind the scenes by these applications. The recycled metadata will be applied to enrich corporate level metadata which can enhance the whole system.

By inputting study level TOC file and corporate level TLG shell metadata into Generate Mock Shell application, study level TLG metadata and mock shell can be generated. Study level TLG metadata like xml file and title & footnote excel file can be served as input material of CDARS application to generate TLG SAS macro parameter values. These macro parameter values will be used to generate SAS program file and final TLG. Post-text TLG can be combined into appendix file of CSR by using Combine TLG application. While Intext TLG will be automatically inserted into CSR with auto-insertion manual generated by CSR Intext Table Insertion application.

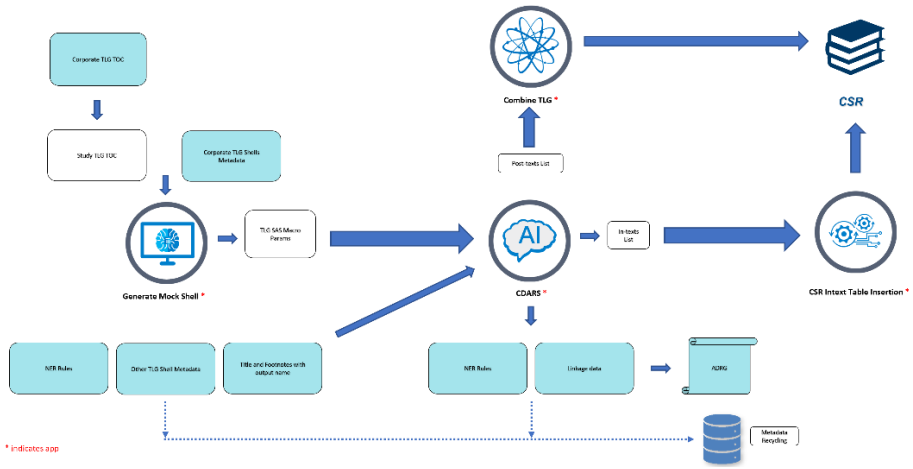


Figure 2. TLG Application Development

GENERATE TLG MOCK SHELL

GLOBAL STANDARD SHELL

Global standard shell is the foundation of this system, and each standard shell is given a unique ID. Figure 3 shows an example table shell - summary of TEAE by System Organ Class and Preferred Term. The shell ID is AE0201.

AE0201
Summary of TEAE by MedDRA System Organ Class and Preferred Term
Safety Analysis Set

System Organ Class Preferred Term	Control (N = xxx) n (%)	Drug 1 (N = xxx) n (%)	Drug 2 (N = xxx) n (%)	Total (N = xxx) n (%)
Subjects with any TEAE	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
System Organ Class 2	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Preferred Term 1	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Preferred Term 2	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Preferred Term 3	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Preferred Term 4	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
***				

Source: ADXX
Data extraction: DDMMMYYYY, Data cut-off: DDMMMYYYY
Abbreviation: N=Number of subjects in the analysis set for each treatment group; TEAE=Treatment-Emergent Adverse Event.
MedDRA version <xx.x> was used for coding.
The percentage was calculated based on N as denominator.
For number of subjects: Subjects with multiple events within a given preferred term and system organ class were counted only once for each preferred term and system organ class, respectively.
</ar-dev/compound-indication/study/.../program name>.sas DDMMMYYYY HH:MM Output file Name

Macro Name: m_ar_ae_socpt

Programming Note:
1. MedDRA version should be updated according to the actual version used in study.
2. The events will be sorted by descending frequency in the <overall/control> group, at SOC and PT level respectively.

Figure 3. Example Standard Shell

CORPORATE LEVEL COLUMN XML AND ROW XML

For each standard shell, we prepared column xml and row xml (see Figure 4) as well as title and footnote excel file. Column headers usually vary from study to study and thus '***[Treat]***' was inserted into column xml to provide flexibility for customization. Sometimes, treatment information is placed in row header like what usually did for laboratory shift table. In this case, '***[Treat]***' will be inserted into row xml file.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<SHELL>
   <TFL id ="1" value1= "" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">AE0201
      <ROWVAR id ="1" value1= "" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">System Organ Class~   Preferred Term</ROWVAR>
      <ROWVAR id ="2" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Subjects with any TEAE</ROWVAR>
      <ROWVAR id ="3" value1= "" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">\~</ROWVAR>
      <ROWVAR id ="4" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">System Organ Class 1
         <LEVEL1 id ="1" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Preferred Term 1</LEVEL1>
         <LEVEL1 id ="2" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Preferred Term 2</LEVEL1>
         <LEVEL1 id ="3" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Preferred Term 3</LEVEL1>
         <LEVEL1 id ="4" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Preferred Term 4</LEVEL1>
      </ROWVAR>
      <ROWVAR id ="5" value1= "" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">\~</ROWVAR>
      <ROWVAR id ="6" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">System Organ Class 2
         <LEVEL1 id ="1" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Preferred Term 1</LEVEL1>
         <LEVEL1 id ="2" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Preferred Term 2</LEVEL1>
         <LEVEL1 id ="3" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Preferred Term 3</LEVEL1>
         <LEVEL1 id ="4" value1= "xx (xx.x)" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">Preferred Term 4</LEVEL1>
      </ROWVAR>
   </TFL>
</SHELL>
```

Row XML File

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<SHELL>
   <TFL id ="1" value1= "" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">AE0201
      <COLUMNVAR id ="1" value1= "" value2= "" value3= "" value4= "" value5= "" value6= " " value7= "" value8= " ">[Treat]~(N = xxx)~n (%)</COLUMNVAR>
   </TFL>
</SHELL>
```

Column XML File

Figure 4. Corporate Level Row XML and Column XML metadata

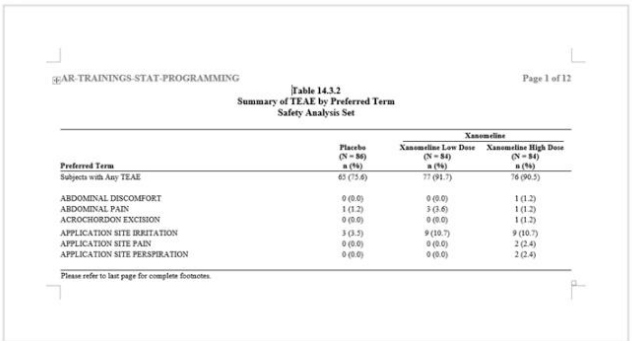
TLG TOC – UNIQUE FOR EACH STUDY

To enable customization of table shell, we established TOC file (left part of Figure 5) which is an excel file consisting of two spreadsheets. The TOC sheet contains type, shell_ID, output number, title, analysis set and outfile name while the Treatment Group sheet includes type, row_header, column_header. The type column and shell ID column can link row_header or column header information with global row xml or global column xml. In this way, study level row xml and column xml can be generated. Multiple column headers allow possibility of defining span column headers. For example, Column_Header1 and Column_Header2 defined for Type B can generate a table containing two levels of columns headers as showed in bottom part of Figure 5.

TOC – Unique for each study

Type	Shell_ID	Output_Number	Title	Analysis_Set	Outfile_Name
A	AE0201	Table 14.3.1	Summary of TEAE by MedDRA System Or	Safety Analysis Set	t-ae-socpt
B	AE0202	Table 14.3.2	Summary of TEAE by Preferred Term	Safety Analysis Set	t-ae-pt
A	LB0101a	Table 14.3.3	Summary of Clinical Chemistry with Chai	Safety Analysis Set	t-lb-chem-chg
C	LB0101b	Table 14.3.4	Summary of Hematology with Change fr	Safety Analysis Set	t-lb-hema-chg

Type	Row_Header	Column_Header1	Column_Header2
A		Placebo	
A		Xanomeline Low Dose	
A		Xanomeline High Dose	
B			Placebo
B		Xanomeline	Low Dose
B		Xanomeline	High Dose
C	Placebo		
C	Xanomeline Low Dose		
C	Xanomeline High Dose		



AR-TRAININGS-STAT-PROGRAMMING Page 1 of 12

Table 14.3.2
Summary of TEAE by Preferred Term
Safety Analysis Set

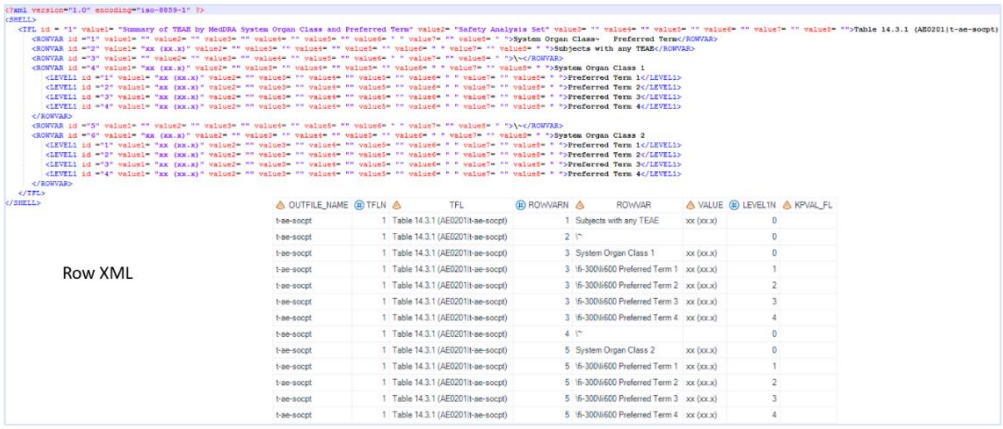
Preferred Term	Placebo (N = 86) n (%)	Xanomeline Low Dose (N = 84) n (%)	Xanomeline High Dose (N = 84) n (%)
Subjects with Any TEAE	65 (75.6)	77 (91.7)	76 (90.5)
ABDOMINAL DISCOMFORT	0 (0.0)	0 (0.0)	1 (1.2)
ABDOMINAL PAIN	1 (1.2)	3 (3.6)	1 (1.2)
ACROCHORDON EXCISION	0 (0.0)	0 (0.0)	1 (1.2)
APPLICATION SITE IRRITATION	3 (3.5)	9 (10.7)	9 (10.7)
APPLICATION SITE PAIN	0 (0.0)	0 (0.0)	2 (2.4)
APPLICATION SITE PERSPIRATION	0 (0.0)	0 (0.0)	2 (2.4)

Please refer to last page for complete footnotes.

Figure 5. TLG TOC

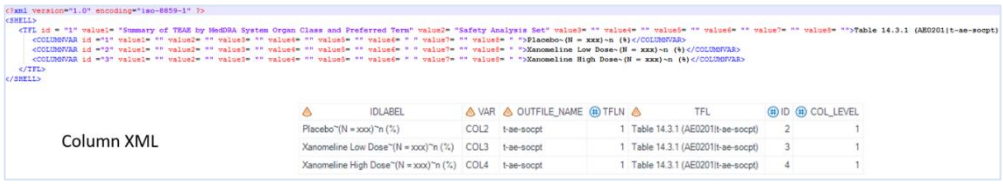
LINK TOC AND XML METADATA TO GENERATE STUDY LEVEL XML METADATA

By linking TOC and coporate level xml, study level row xml (Figure 6) and column xml (Figure 7) can be generated. it is not hard to find that [Treat] from coporate level XML has been replaced with real column header, Placebo, Xanomeline Low Dose and Xanomeline High Dose. These xml files can be converted into SAS dataset (Figure 6 and Figure 7).



OUTFILE_NAME	TFLN	TFL	ROWVARN	ROWVAR	VALUE	LEVEL1N	KPVAL_FL
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	1	Subjects with any TEAE	xx (xx.x)	0	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	2	\"		0	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	3	System Organ Class 1	xx (xx.x)	0	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	3	\fi-300\li600 Preferred Term 1	xx (xx.x)	1	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	3	\fi-300\li600 Preferred Term 2	xx (xx.x)	2	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	3	\fi-300\li600 Preferred Term 3	xx (xx.x)	3	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	3	\fi-300\li600 Preferred Term 4	xx (xx.x)	4	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	4	\"		0	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	5	System Organ Class 2	xx (xx.x)	0	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	5	\fi-300\li600 Preferred Term 1	xx (xx.x)	1	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	5	\fi-300\li600 Preferred Term 2	xx (xx.x)	2	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	5	\fi-300\li600 Preferred Term 3	xx (xx.x)	3	
t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	5	\fi-300\li600 Preferred Term 4	xx (xx.x)	4	

Figure 6. Study Level Row XML and Corresponding SAS dataset



IDLABEL	VAR	OUTFILE_NAME	TFLN	TFL	ID	COL_LEVEL
Placebo~(N = xxx)~n (%)	COL2	t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	2	1
Xanomeline Low Dose~(N = xxx)~n (%)	COL3	t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	3	1
Xanomeline High Dose~(N = xxx)~n (%)	COL4	t-ae-socpt	1	Table 14.3.1 (AE0201\t-ae-socpt)	4	1

Figure 7. Study Level Column XML and corresponding SAS dataset

GENERATE FINAL RTF OUTPUT

By using cross join [1] from PROC SQL and PROC TRANSPOSE procedure, SAS datasets can be merged into a new dataset which will be used by PROC REPORT to produce TLG mock shell.

```
proc sql;
create table table as select a.*, b.id, b.idlabel, b.col_level
    from row as a cross join column as b
        where a.tfln = b.tfln and a.tfl = b.tfl
            order by tfln, tfl, rowvarn ,level1n ,rowvar, col_level, id;quit;

proc transpose data=table out=t_ae_socpt(drop=_name_ rename=(rowvar = col1)) prefix=col;
    by tfln tfl rowvarn level1n rowvar col_level;
    idlabel idlabel;
    id id;
    var value;
run;
```

TFLN	TFL	ROWVARN	LEVEL1N	COL1	COL_LEVEL	COL2	COL3	COL4
1	Table 14.3.1 (AE0201\t-ae-socpt)	1	0	Subjects with any TEAE	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	2	0	\"	1			
1	Table 14.3.1 (AE0201\t-ae-socpt)	3	0	System Organ Class 1	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	3	1	\fi-300\li600 Preferred Term 1	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	3	2	\fi-300\li600 Preferred Term 2	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	3	3	\fi-300\li600 Preferred Term 3	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	3	4	\fi-300\li600 Preferred Term 4	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	4	0	\"	1			
1	Table 14.3.1 (AE0201\t-ae-socpt)	5	0	System Organ Class 2	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	5	1	\fi-300\li600 Preferred Term 1	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	5	2	\fi-300\li600 Preferred Term 2	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	5	3	\fi-300\li600 Preferred Term 3	1	xx (xx.x)	xx (xx.x)	xx (xx.x)
1	Table 14.3.1 (AE0201\t-ae-socpt)	5	4	\fi-300\li600 Preferred Term 4	1	xx (xx.x)	xx (xx.x)	xx (xx.x)

Figure 8. Cross Join Row and Column Dataset and Transpose for Final Output Dataset

GENERATE MOCK SHELL APPLICATION

To further meet the more diversified customization needs of users while minimize the time for generating and modifying TLG shells, an application was designed as showed in Figure 9. The left part shows steps that users must go through to generate TLG shell in RTF format.

- **Step 1**: Select study level TOC file. Button ‘Generate Outfile_Name’ can generate outfile name for each TLG using NLP (Nature Language Process) while button ‘Edit Outfile_Name’ can help update the outfile name.
- **Step 2**: By clicking ‘Generate Initial XML’ button, two xml files containing row and column information will be created automatically. Button ‘Update Individual XML’ can activate TOC tab. The third button ‘Combine Individual XML’ can bundle all induvial xml files together.

- **Step 3**: By clicking buttons ‘Row xml2tree’ and ‘Column xml2tree’, overall xml files will be displayed as ‘row tree’ and ‘column tree’ in right part of GUI. Right-click menu feature as well as Delete undo/redo is provided for customization of shell. Buttons ‘Row tree2xml’ and ‘Column tree2xml’ can allow us to convert customized tree into xml files.
- **Step 4**: Initial title and footnote excel file will be generated based on global footnote metadata once ‘Generate Title and Footnote Excel’ is clicked. A button was also created to open this excel file for customization.
- **Step 5**: Row xml file, column xml file, title and footnote metadata file will be used to create RTF file for each TLG by clicking on ‘XML2RTF’. Button ‘Combine RTF’ enable us to bundle individual rtf files into final TLG shell.

Repeat step 2 or 3, step 4 (no need to generate initial title and footnote metadata) and step 5 to update TLG shell once get comments from Biostatistician, Medical and other functions.

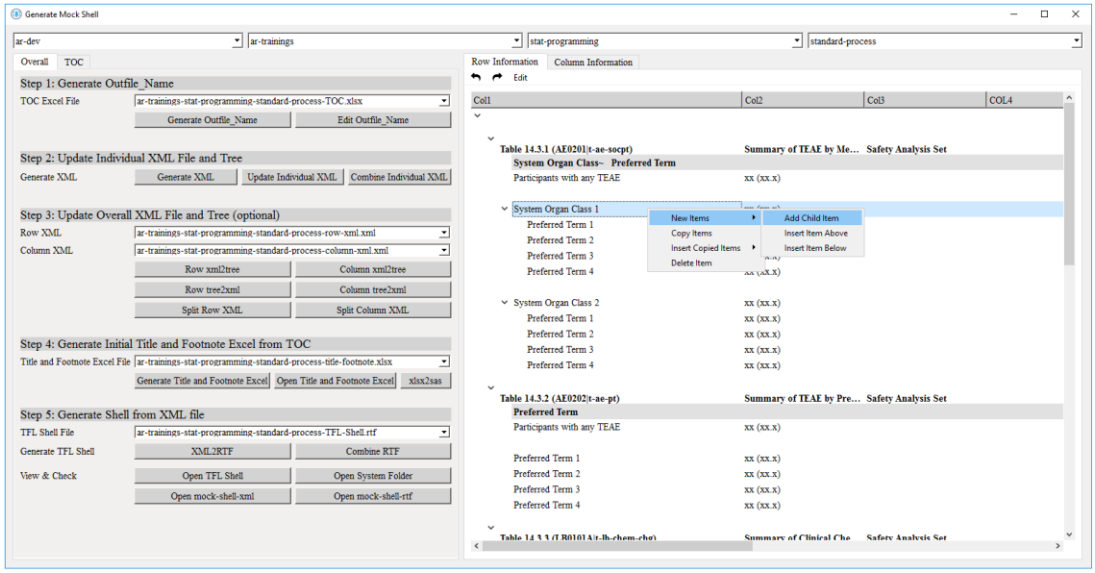


Figure 9. Generate Mock Shell Application

To make it easier to modify xml files for just one TLG, a TOC tab (see Figure 10) was also generated to contain TLG information organizing in the same way as it is stored in TLG TOC file. Right click menu on specific Outfile_Name cell can help generate initial xml file, convert xml files into row/column tree, generate RTF and even delete xml files.

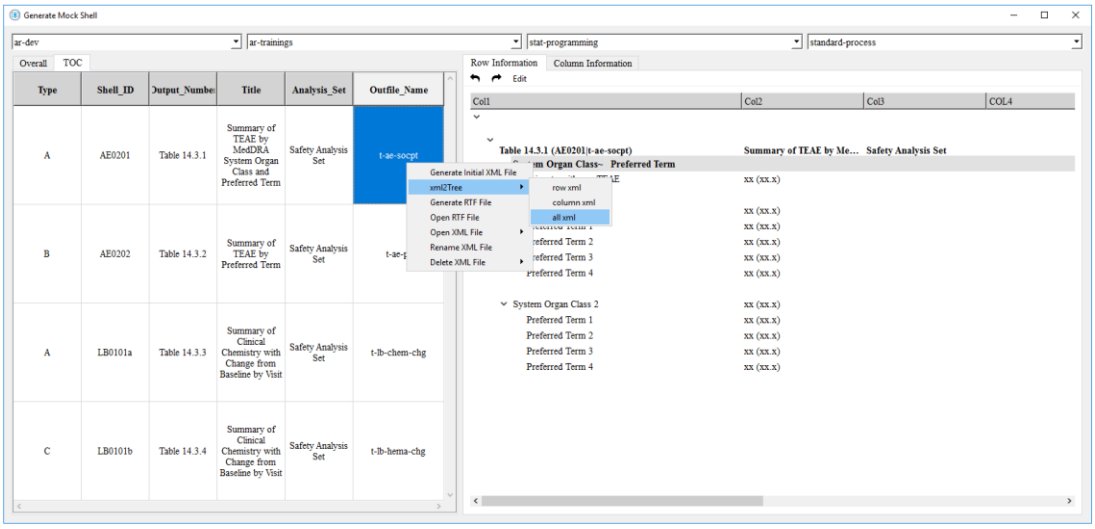


Figure 10. TOC tab of Generate Mock Shell Application

GENERATE TLG OUTPUT

There are two types of TLG, TLG requiring many derivations and those require no or few derivations. Automatic generation of first type relies heavily on SAS reporting macros while creation of second type heavily depends on metadata file linking columns and variables together.

TLG REQUIRING MANY DERIVATIONS

SAS Reporting Macro System

Besides xml metadata and footnote metadata, SAS reporting macros were also generated based on global standard TLG mockup and Shell ID (Figure 11). For example, SAS macro m_ar_ae_socpt was intended to develop AE summary table having shell ID like AE0201, AE0202, AE0301 and AE0302.

These reporting SAS macros have same prefix 'm_ar_' and developed based on same unit macros (e.g. u_lbl2header), analysis macros (e.g. m_ar_count and m_ar_mean) and report macro m_ar_reports. For efficacy TLG, SAS procedures instead of m_ar_count and m_ar_mean will be used to increase readability and understandability for reviewers.

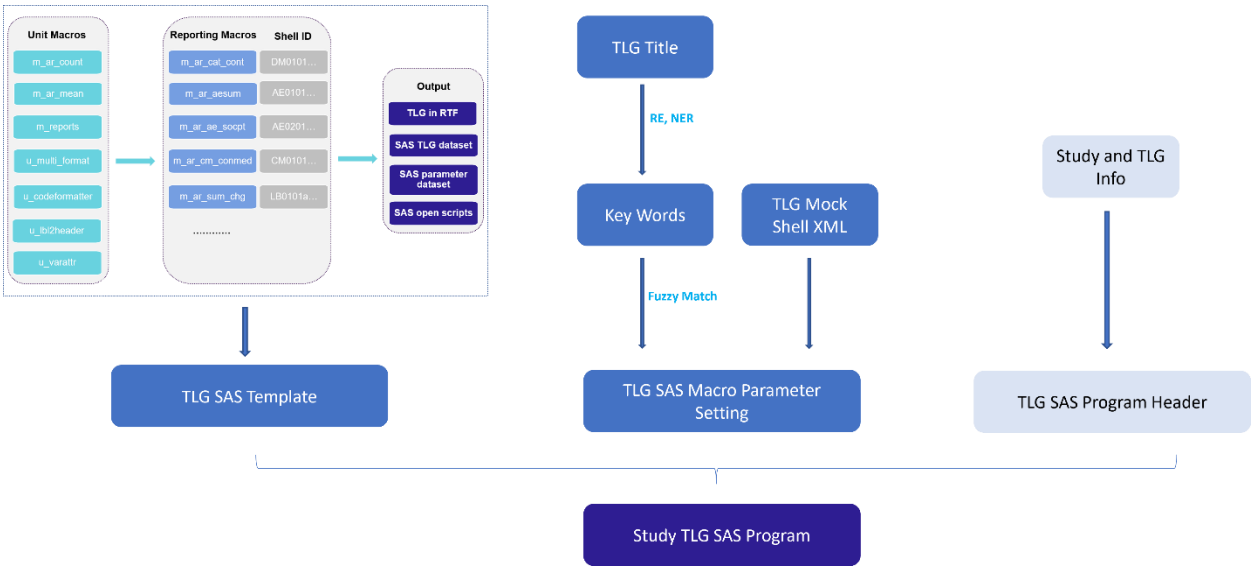


Figure 11. SAS Reporting Macro System

Outputs generated by these reporting macros include RTF file, SAS TLG output dataset, SAS parameter dataset and SAS open code. Main function of SAS open scripts is to be submitted to regulatory authorities. It can generate same TLG as parent SAS program file does. Moreover, only few sub macros like m_ar_count, m_ar_mean and m_reports can be found in these open scripts. Comments regarding sub macro feature and parameter usage will be added right before the code calling these sub macros. With good formatting and comments, it is very easy for reviewer to understand the code.

Multi-language is also handled by these macros. By switching value of parameter LANGUAGE, users can produce both English and Chinese TLG. Fixed wording will be translated directly by macros. Table content coming from ADaM or SDTM dataset should be translated at ADaM or SDTM level.

TLG SAS TEMPLATE

Given the fact that there is a one-to-one or one-to-many relationship between SAS reporting macro and shell ID, it is very easy to generate a SAS program template file for each TLG. Figure 12 presents a template file for table with shell ID AE0201. It consists of program header part and SAS macro part.

Information required by program header and macro parameter values can vary from table to table. To make it easy for automation, they are broken into small units enclosed in brackets. These small units can be replaced with real code or text customized for a specific TLG so that SAS program file to create this TLG output can be generated.

```
/*-------------------------------------------------------------------------------
Study:              [Protocol]
Program:            [program_name].sas
Keywords:           TEAE, SOC, PT
Function:           [title]
Author:             [username]
Date Completed:     [date]
SAS Version:        9.4
Platform:           SUSE Linux Enterprise Server 15
Input Data:         ADSL, ADAE
Macros Called:      m_ar_ae_socpt
Program Output:     [outfile_name].rtf
Program log:        [outfile_name].log
Macro Parameters:   None

Program Flow:
   Step 01:
   Step 02:

Limitations/Cautions:

Revisions:

-------------------------------------------------------------------------------*/

%m_ar_ae_socpt(
 input_dataset           = lptad.adae
,input_dataset_where     = saffl = 'Y'[teaeCode][tesaeCode][saeCode][relatedCode][grd3Code][leadingCode]
,description_text        = Subjects with Any[relatedText][teaeText][tesaeText][saeText][grd3Text][leadingText]
,trt_var                 = trta
,level1_var              = [lvl1Code]
,level2_var              = [lvl2Code]
,level3_var              = [lvl3Code]
,population_dataset      = lptad.adsl
,population_dataset_where= saffl = 'Y'
,therapy_var             =
,display_total           =
,display_trt             =
,grp_var                 = [grpCode]
,display_grp             = [grpText]
,level1_sortby           =
,level2_sortby           =
,sortby_colnum           =
,bign_type               =
,cutoff                  = [pctCode]
,report_yn               =
,page_vars               =
,rpt_span_headers        = [rpt_span_headers]
,column_width            =
,col1_header             =
,line_num_per_page       =
,outfile_name            = [outfile_name]
,title_footnote_template = lptmtsy.tlaft
,tfl_title               =
,tfl_footnote            =
,footnote_on_last_page   =
,output_path             =
,output_pgm_path         = [output_pgm_path]
,output_datlib           =
,output_dataset          =
,layout_orientation      =
,font_size               =
,language                = [language]
,destination             = [destination]
,help                    =
,debug                   =
);
```

Figure 12. SAS Program Template

EXTRACT INFORMATION FOR TLG TITLE AND XML FILES

It is not hard to get information like protocol number, program name, TLG title, program author and substitute them for small units in program template file. But for other small units like '[teaeCode]', '[tesaeCode]' and '[lvl1Code]', the only way to get related information is to check TLG title and mock shell (xml files in our case). At Dizal, we applied regular expression and rule-based matching of Spacy package to extract Named-entity recognition (NER) which can be used to derive small units. Below shows the code regarding how to perform NER for AE summary table title [2].

The extracted group of words by Spacy package were assigned labels like byGroup, leadingGroup, relatedGroup, pctGroup, cycGroup, teaeGroup and so on. For title '*Summary of TEAE by MedDRA Systerm Organ Class and Preferred Term*', we can get three NERs which are '***TEAE***', '***MedDRA System Organ Class***' and '***Preferred Term***'. Their entity names are ***teaeGroup, lvl1Group and lvl2Group***.

```
title = 'Summary of TEAE by MedDRA System Organ Class and Preferred Term'

ae_expression_dict = {
'relatedGroup': r"(\w+-related|\w+-induced)",
'pctGroup': r"(((\^|\(\*ESC)\D+\d{4,}\D+|\W+)\s*\d+\s*(percent|pct|%))",
'grd3Group': r"(grade 3 or higher|grade\s+\W+\s*\d)",
'cycGroup': r"(Cycle\s\d+\sor\sCycle\s\d+\sDay\s\d+|Cycle\s\d+\sDay\s\d+|Cycle\s\d+)|(Day\s\d+)",
'teaeGroup': r"(Treatment Emergent Adverse Event|Treatment-Emergent Adverse Event|Treatment Emergent
AE|Treatment-Emergent AE|TEAE|TEAEs)",
'tesaeGroup': r"(Treatment Emergent Serious Adverse Event|Treatment Emergent Serious AE|TESAE|TESAEs)",
'saeGroup': r"(Serious Adverse Event|SAE)",
'aeGroup': r"(adverse event|AE)"
                    }

ner_text = []
ner_name = []


# Regex
title1 = title
for key in ae_expression_dict:
    match_lst = re.findall(ae_expression_dict[key], title1, flags=re.IGNORECASE)
    if match_lst is not None:
        for match in match_lst:
            if isinstance(match, str) == True:
                match = match
            else:
                match = match[0]
            ner_text.append(match.strip())
            ner_name.append(key)
            title1 = title1.replace(match, "")
            title1 = re.sub(' +', ' ', title1)
```

```
nlp = spacy.load("en_core_web_sm", disable=['ner'])
# Entity for by Group
bygroup_pattern = [
    {"LOWER":"by"},
    {"POS":"{}", "OP": "*"},
    {"POS": {'NOT_IN': ['ADP', 'PUNCT']}, "OP":"*"},
    {"TEXT": {'REGEX': '(?i)(by|,|of)'}, "OP": "?"},
    {"POS": {'NOT_IN': ['ADP', 'PUNCT']}, "OP": "*"},
    {"TEXT": {'REGEX': '(?i)(by|,)'}, "OP": "?"},
    {"POS": {'NOT_IN': ['ADP', 'PUNCT']}, "OP": "*"},
    {"POS": {'NOT_IN': ['ADP', 'VERB', 'PART', 'CCONJ', 'PUNCT']}}
]
patterns_bygroup = [{"label":"byGroup", "pattern":bygroup_pattern}]
ruler_bygroup = EntityRuler(nlp, overwrite_ents=True)
ruler_bygroup.add_patterns(patterns_bygroup)
ruler_bygroup.name = 'ruler_bygroup'

# Entity for leading to Group
leadingGroup_pattern = [
    {"LOWER": "leading"},
    {"LOWER": "to"},
    {"TEXT": {'REGEX': '(?i)(death)'}, "OP": "?"},
    {"POS": {'NOT_IN': ['ADP']}, "OP": "?"},
    {"TEXT": {'REGEX': '(?i)(of|from)'}, "OP": "?"},
    {"POS": {'NOT_IN': ['ADP']}, "OP": "*"},
    {"POS": {'NOT_IN': ['ADP', 'VERB', 'PART', 'CCONJ']}}
]
patterns_leading = [{"label":"leadingGroup", "pattern":leadingGroup_pattern}]
ruler_leading = EntityRuler(nlp, overwrite_ents=True)
ruler_leading.add_patterns(patterns_leading)
ruler_leading.name = 'ruler_leading'

……

nlp.add_pipe(ruler_bygroup)
nlp.add_pipe(ruler_leading)

# Entity Ruler
doc =nlp(title1)
for ent in doc.ents:
    ner_text.append(ent.text.strip())
    ner_name.append(ent.label_)
```

Figure 13. Code to Extract Key Words from TLG Title

Macro Mapping Specification

With the macro mapping specification showed in Figure 14 and fuzzy match of fuzzywuzzy package, the extracted text and labels can be derived as small units. For example, label 'teaeGroup' can derive two small units - '[teaeCode]' and '[teaeText]'. Their respective values are ***and strip(upcase(trtemfl)) = 'Y'*** and ***TEAE***. Label 'lvl1Group' and text 'MedDRA System Organ Class' can get small unit '[lvl1Code]' and corresponding value is '***aebodsys***'. While label 'lvl2Group' and text 'Preferred Term' can get value of '[lvl2Code]' assigned as 'aedecod'.

Ner_Name	Mapping_Text	Code	Text
lvl1Group	System Organ Class	aebodsys	
lvl1Group	Standard MedDRA Query	smq01nam smq02nam smq03nam smq04nam smq05nam	
lvl1Group	customized Query	cq01nam cq02nam cq03nam cq04nam cq05nam cq06nam	
lvl1Group	Grouped Preferred Term	aeptgr	
lvl1Group	Preferred Term	aedecod	
lvl2Group	Grouped Preferred Term	aeptgr	
lvl2Group	Preferred Term	aedecod	
lvl3Group	Maximum CTCAE Grade	atoxgr|atoxgrn	
lvl3Group	Worst Grade	atoxgr|atoxgrn	
grpGroup	CTCAE Grade	atoxgr	
grpGroup	Grade	atoxgr	
grpGroup	Treatment		
relatedGroup	Drug-Related	and strip(upcase(aerel)) = 'RELATED'	drug-related
grd3Group	Grade 3 or Higher	and atoxgrn >= 3	Grade 3 or Higher
grd3Group	Grade >= 3	and atoxgrn >= 3	Grade 3 or Higher
leadingGroup	Leading to Death	and strip(upcase(aesdth)) = 'Y'	Leading to Death
leadingGroup	Leading to Discontinuation from Study	and strip(upcase(aeacn)) = ""	Leading to Discontinuation from Study
leadingGroup	Leading to Treatment Discontinuation	and strip(upcase(aeacn)) = 'DRUG WITHDRAWN'	Leading to Treatment Discontinuation
leadingGroup	Leading to Treatment Interruption	and strip(upcase(aeacn)) = 'DRUG INTERRUPTED'	Leading to Treatment Interruption
leadingGroup	Leading to Treatment Reduction	and strip(upcase(aeacn)) = 'DOSE REDUCED'	Leading to Treatment Reduction
leadingGroup	Leading to Withdrawal from Study	and strip(upcase(aeacn)) = ""	Leading to Withdrawal from Study
teaeGroup	Treatment Emergent Adverse Event	and trtemfl = 'Y'	TEAE
teaeGroup	Treatment Emergent AE	and trtemfl = 'Y'	TEAE
teaeGroup	TEAE	and trtemfl = 'Y'	TEAE
tesaeGroup	Treatment Emergent Serious Adverse Event	and trtemfl = 'Y' and strip(upcase(aeser)) = 'Y'	TESAE
tesaeGroup	TESAE	and trtemfl = 'Y' and strip(upcase(aeser)) = 'Y'	TESAE
saeGroup	Serious Adverse Event	and strip(upcase(aeser)) = 'Y'	SAE
saeGroup	SAE	and strip(upcase(aeser)) = 'Y'	SAE
aeGroup	Adverse Event		AE

Figure 14. Macro Mapping Specification

By replacing these small units with corresponding values, SAS program template file can be customized for a specific TLG. Those small units not being able to be extracted from TLG title will be replaced with blank space.

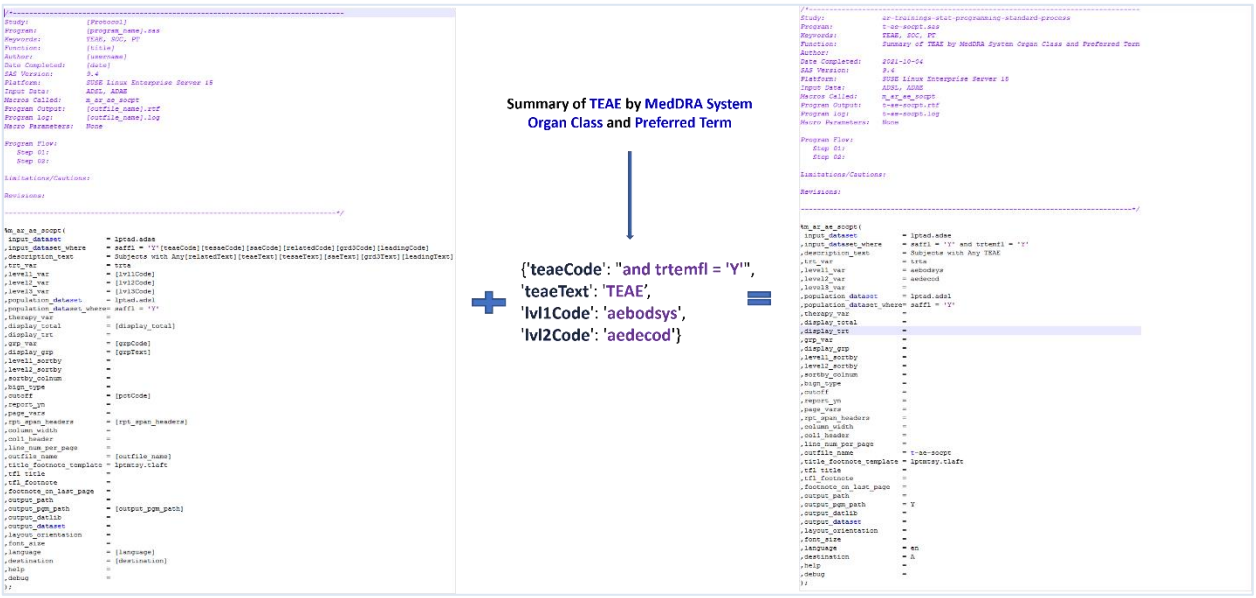


Figure 15. Example of Automatically Generated TLG SAS Program

LISTING REQUIRE NO OR FEW DERIVATIONS

Listing Metadata and Auto-Mapping

A metadata file containing column header, sub header will be generated from mock shell XML file. With Shell_ID, Col_Header, Sub_Col_Header, columns like Source_Dataset, Source_Variable, Data_Type, and Format can be filled by merging with global metadata.

Shell_ ID	Output_ Type	Output_ Number	Title1	Population	Outfile_ Name	SEQ	Col_Header	Sub_SEQ	Sub_Col_ Header	Source_Dataset	Main _YN	Source_ Variable	Data _Typ	Format	Codelist	Sort_ YN	Group_ YN	Line_Brk_ YN	Column _Width
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	1	Participant ID	1	Participant ID	LPTAD.ADAE(where =(saffl = "Y"))		SUBJID	Char			Y	Y	Y	9
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	2	Sex/Age~(years)/~Race	1	Sex			SEX	Char				Y		9
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	2	Sex/Age~(years)/~Race	2	Age (years)			AGE	Integer						9
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	2	Sex/Age~(years)/~Race	3	Race			RACE	Char						9
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	3	Term/~Preferred Term/~System Organ~Class	1	AE Term			AETERM	Char						9
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	3	Term/~Preferred Term/~System Organ~Class	2	Preferred Term			AEDECOD	Char						9
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	3	Term/~Preferred Term/~System Organ~Class	3	System Organ Class			AEBODSYS	Char						9
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	4	(Day)/~End Date (Day) ("ESC")|super	1	Start Date (Day)			AESTDTC. ADY	Char	YYMMDD10.		Y			9
AE9901	L	16.2.7.1	All Adverse Events	Safety Analysis Set	l-ae	4	(Day)/~End Date (Day) ("ESC")|super	2	End Date (Day)			AEENDTC. ADY	Char	YYMMDD10.		Y			9

Figure 16. Example of Listing Metadata

For listings not covered by corporate level mock shell, source dataset and variables as well as datatype, format can be populated by finding most similar label from ADaM and SDTM spec comparing against column header using NLP. Below presents code about how to map source variables.

```
import spacy
import re
import numpy as np
import pandas as pd

spec_df  ……..
lst_df   ……..

nlp = spacy.load('en_core_web_lg')
label_lst = spec_df['LABEL'].tolist()

for index, row in lst_df.iterrows():
    col_header = str(row["SUB_COL_HEADER"])
    col_header = re.sub('{.+}', '', col_header)
    col_header = re.sub('\(.+\)', '', col_header)
    search_txt = nlp(col_header)

    sims= []
    for lbl in nlp.pipe(label_lst):
        sims.append(search_txt.similarity(lbl))

    id_max = np.argmax(sims)
    matched_label = label_lst[id_max]

    if sims[id_max] > 0.6:
        matched_row = spec_df.loc[spec_df['LABEL'] == matched_label,]
        matched_variable = matched_row['VARIABLE'].values[0]
        matched_datatype = matched_row['DATATYPE'].values[0]
        matched_format = matched_row['FORMAT'].values[0]

        lst_df.loc[index, 'SOURCE_VARIABLE'] = matched_variable
        lst_df.loc[index, 'DATA_TYPE'] = matched_datatype
        lst_df.loc[index, 'FORMAT'] = matched_format

        label_lst.remove(matched_label)
    else:
        lst_df.loc[index, 'SOURCE_VARIABLE'] = ''
        lst_df.loc[index, 'DATA_TYPE'] = ''
        lst_df.loc[index, 'FORMAT'] = ''
```

Figure 17. Code to Map Source Variable with Column Header

Convert Listing Metadata into Program File

After manually check and update the metadata, listing metadata file can be converted into SAS program file by calling SAS macro u_shell_meta2code. Below shows example metadata converted program file. The code is well structured. Even if required source information is not filled, similar code can also be generated. Users only need to update Process input dataset section and thus still saves a lot of time for statistical programmers.

```
/*-----------------------------------------------------------------------------
Study:              ar-trainings-stat-programming-standard-process
Program:            l-ae.sas
Keywords:           l-ae
Function:           This program is used to generate Listing 16.2.7.1 All Adverse Events (core)
Author:
Limitations/Cautions:
Revisions:
-----------------------------------------------------------------------------*/

*****************************************************************************************************;
*** Read input dataset;
*****************************************************************************************************;
data indat1;
   set lptad.adae(where=(saffl = "Y"));
run;

*****************************************************************************************************;
*** Process input dataset;
*****************************************************************************************************;
data indat2;
   length col1 col10 col11 col12 col2 col6 col7 col8 col9 col31 col32 col33 col41 col42 col43 col51 col52 col53 $200.col3 $600. col4 $600. col5 $600. ;
   set indat1;
   ***col1: [Cohort/~Treatment]***;
   col1="" ;

   ***col2: Participant~ID***;
   col2 = strip(subjid);

   ***col3: Sex/Age~(years)/Race***;
   col31 = strip(sex);
   if missing(age) = 0 then col32 = strip(put(age, best.));
   else col32 = ' ';
   col33 = strip(race);
   col3= catx('/\line ',strip(col31), strip(col32), strip(col33));

run;

*****************************************************************************************************;
*** Derive final dataset;
*****************************************************************************************************;
data final;
   set indat2;
run;

proc sort data=final; by aestdtc aeendtc; run;
*****************************************************************************************************;
*** TFL title and footnote;
*****************************************************************************************************;
data tlaft;
   SHELL_ID = "AE9901S" ;
   OUTPUT_TYPE = "L" ;
   SECTION_TITLE1 = "" ;
   SECTION_TITLE2 = "" ;
   OUTPUT_NUMBER = "16.2.7.1" ;
   TITLE1 = "All Adverse Events (core)" ;

run;

*****************************************************************************************************;
*** Macro purpose:  Construct PROC REPORT and generate RTF file
*****************************************************************************************************;
%m_reports(
 input_dataset           =  final
,rpt_vars                =  col1 col2 col3 col4 aestdtc aeendtc col5 col6 col7 col8 col9 col10 col11 col12
,rpt_headers             =  %nrbquote([ [Cohort/~Treatment]| Participant~ID| Sex/Age~(years)/Race| AE Term/~Preferred Term/~System Organ~Class (*ESC*){super a}|
,column_width            =  5 5 9 9 5 5 5 5 9 9 5 5
,column_alignment        =  l l l l l l l l l l l l
,header_alignment        =
,sort_vars               =  aestdtc aeendtc
,noprint_vars            =  aestdtc aeendtc

,outfile_name            =  l-ae-pa
,title_footnote_template =  tlaft

,language                =  EN
,destination             =  A
,help                    =  N
,debug                   =  N
);
```

Figure 18. Example Listing Program File Created from Metadata

CDARS APPLICATION

To improve working efficiency, we integrated the process of reviewing mockup, generating program file, reviewing program file, submitting program and opening RTF output into one application – CDARS.

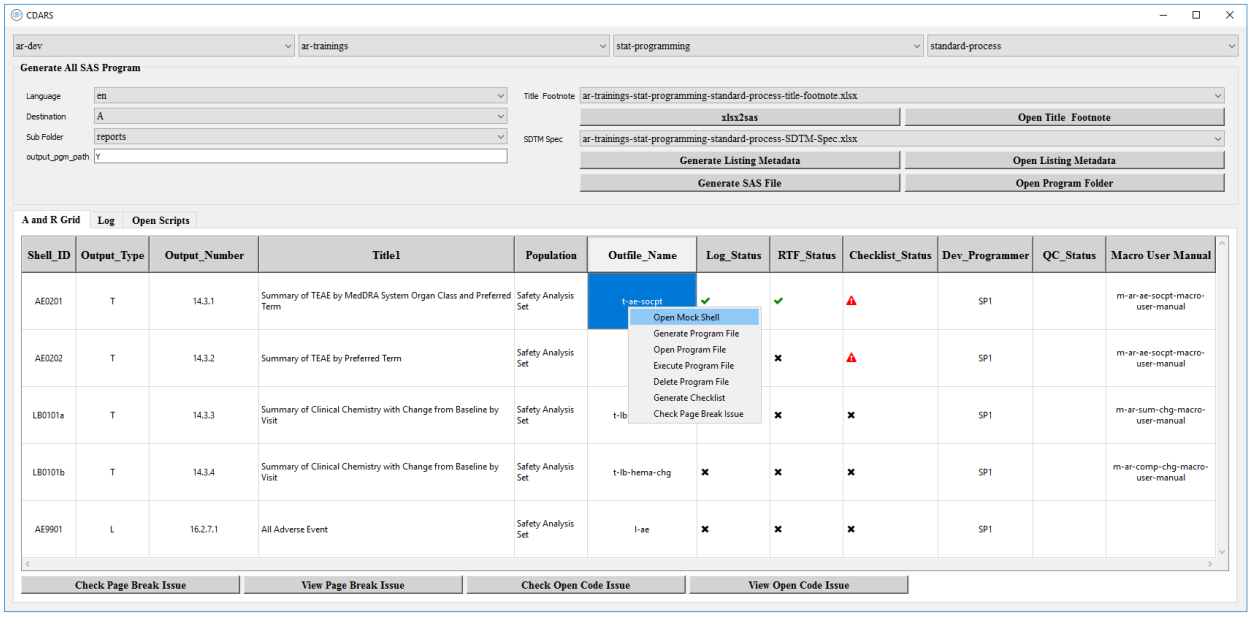


Figure 19. CDARS Application

By selecting title and footnote excel, Shell_ID, Output_Number, Table Title, Outfile_Name and other information will be extracted and displayed in table as showed in Figure 19. Log status, RTF status and Checklist status can also be displayed. Reviewing QC status can be completed by pressing right click on column header of 'QC_Status' column.

By clicking on 'Generate SAS Program File', initial SAS program containing program header and macro code will be generated for each TFL based on Shell_ID. Already existed program file will not be over-written. By clicking on cells under 'Log_Status', 'RTF_Status' and 'Checklist_Status' columns, corresponding files can be opened for review.

Another two feature about page break issue and open code issue is also integrated.

CONCLUSION

This paper introduced how to achieve the task of TLG Auto Generation at Dizal. This process has been used by several studies and has been proved to have a great ability to increase efficiency. With more and more data recycled from studies, we believe that the productivity can be further improved.

REFERENCES

1. Chenggeng Tian. "*Decomposition and Reconstruction of TLF Shells - A Simple, Fast and Accurate Shell Designer*" PharmaSUG China 2018.

2. Sumesh Kalappurakal, Srikanth Ramakrishnan, Shobha Rani, Harry Chen. "*Metadata-based Auto-Programming Process – Part 3: Virtual AI assistant to automate TLG generation*" PhUSE US Connect 2020

ACKNOWLEDGMENTS

I would like to thank Huadan Li and all statistical programming team members at Dizal for their advice and feedback so that the process can be improved continuously.

CONTACT INFORMATION

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

Zhiping Yan
Dizal Pharma
+86-13691423469
zhiping.yan@dizalpharma.com