PharmaSUG 2018 - Paper SI-11

The Next Generation Smart Program Repository

Hrideep Antony, Syneos Health; Aman Bahl, Syneos Health

ABSTRACT

The term repository sounds very “routine”, but what if it can do the job of both a “project manager” and a “SAS® programmer”!

That is right! Introducing the smart repository that will create initial SAS® Programs, generate status updates, track project progress, create audit trails, and check program logs, all with the click of a few buttons!

While this utility provides a structured approach to programming and increases the programming efficiency, it also ensures the compliance of each program, making them audit ready. Combined features of excel VBA and macros are used to build this utility.

INTRODUCTION

This utility consists of three key features:

- The excel VBA front-end screen, which acts as an interface and controls the overall repository;
- The status summary screen, which provides instant and customizable summary reports of project progress; and
- The SAS® utility macros, which generate programs and auto-populate program headers, based on information provided in the VBA screen.

Figure 1 below provides insights into the overall process flow of this utility.

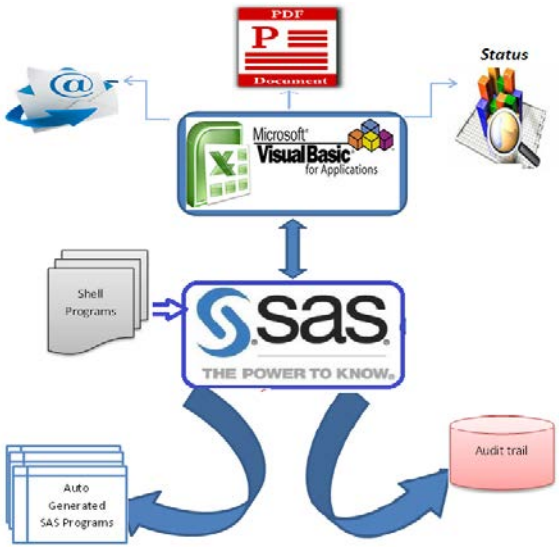

Figure 1: Process Flow Chart

The repository uses powerful features of excel VBA to control and generate instantaneous overall status reports. VBA macros ensure the overall quality of the repository by prompting the user with warnings, messages and color coding, for the issues discovered when the “submit” button is clicked by the user.

The SAS® utility macros, once initiated, create new SAS® programs based on the file and folder locations defined in the repository. The macros also support the accuracy of the program location, as well as ensure the existing ones are not replaced. The newly created SAS® programs will have auto-populated headers with output locations, output path names, and other relevant information.

This structured automated approach to programming and tracking increases the overall efficiency of the program development cycle, while delivering the most up-to-date status information with minimal effort. This paper will further describe various functional aspects of this utility.

EXCEL VBA FRONT- END SCREEN

Figure 2 below shows a screenshot of the VBA front-end screen, which acts as the front end of this repository.

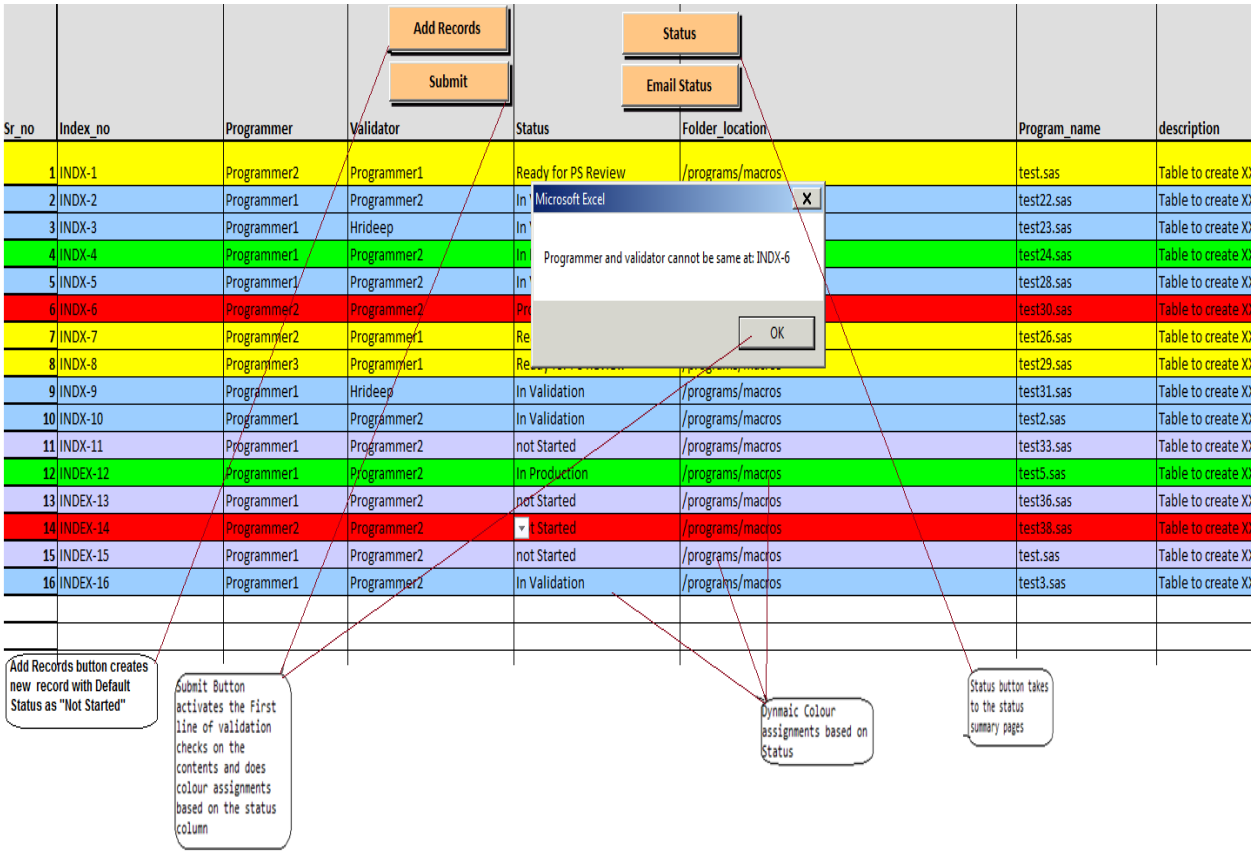

Figure 2: Excel VBA Front-End Screen

The “**Submit**” button activates validation checks such as missing program names, and same programmer and validator assignments(as both programmer and validator cannot be the same person). The “submit” button also assigns a color to each record based on the status. Once the validation checks are complete, the SAS® macro, that generates the SAS® shell programs, is initiated. The SAS® program uses the program location cited in the “folder_location” column, and generates programs with the SAS® program names cited under the column “Program_name”.

The below VBA code is initiated when the “submit” button is clicked. This code will perform several basic validation checks and color coding.

```
Private Sub submit_button()
 'defining and assigning values to the variables
     Dim i As Long, r1 As Range, r2 As Range, r3 As Range, r4 As Range
' defining the loop and assign cell values to variables to check the validity of data
   For i = 2 To 1000
      Set r1 = Range("E" & i)
      Set r2 = Range("A" & i & ":O" & i)
      Set r3 = Range("C" & i)
      Set r4 = Range("D" & i)
      ' Assign colour codes based on the status of each cell
      If r1.Value = "In Validation" Then r2.Interior.ColorIndex = 37
      If r1.Value = "In Production" Then r2.Interior.ColorIndex = 4
      If r1.Value = "not Started" Then r2.Interior.ColorIndex = 24
            If r1.Value = "Ready for PS Review" Then r2.Interior.Color = vbYellow
            If r1.Value = "In Programming" Then r2.Interior.ColorIndex = 16
      ' if key variables are missing then assign colour code red and prompt a message box to display the error
             If Range("A" & i).Value > 0 And Range("G" & i).Value = "" Then
               r2.Interior.Color = vbRed
               cn = ActiveCell.Column
               cr = Range("B" & i).Value
               MsgBox ("Program Name cannot be missing at: " & cr)
             End If

             If Range("C" & i).Value = Range("D" & i).Value And Range("A" & i).Value > 0 Then
               crn = Range("B" & i).Value
               r2.Interior.Color = vbRed
               MsgBox ("Programmer and validator cannot be same at: " & crn)
             End If
        Next i

 End Sub
```

The “**Add records**” button creates a new record by automatically incrementing the index numbers and setting the status defaults to “Not started”. The below code is initiated when the “add records” button is clicked.

```
Private Sub add_record_Click()
'Add new records with common information from the previous records
 If WorksheetName = vbNullString Then WorksheetName = ActiveSheet.Name
 With Worksheets(WorksheetName)
  xllastrow = .Cells.Find("*", .Cells(1), xlFormulas, _
      xlWhole, xlByRows, xlPrevious).Row
      Rows(xllastrow).Copy
    Range("A" & xllastrow + 1).Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
    Range("B" & xllastrow + 1).Value = "INDEX-" & Range("A" & xllastrow).Value + 1
    Range("A" & xllastrow + 1).Value = Range("A" & xllastrow).Value + 1
    'Set default values to "not started"
    Range("D" & xllastrow + 1).Value = "not Started"
    Range("E" & xllastrow + 1).Value = "not Started"
    Range("G" & xllastrow + 1).Value = ""
End With
End Sub
```

EXCEL VBA STATUS SUMMARY SCREEN

The "**Status**" button will initiate a summary status page as shown in Figure 3. Below you will notice the graphics are highly customizable, and a number of filter options have been incorporated. The VBA code used here is to open the status tab and refresh the summary status.

```
Private Sub Status_button()
'Initiate the "Status" tab and refresh the tables based on the current information in the main sheet
  ThisWorkbook.Sheets("status").Activate
  Sheets("status").PivotTables("PivotTable2").PivotCache.Refresh
  ThisWorkbook.Sheets("status").Activate
  Sheets("status").PivotTables("PivotTable3").PivotCache.Refresh
  ThisWorkbook.Sheets("status").Activate
  Sheets("status").PivotTables("PivotTable4").PivotCache.Refresh
End Sub
```

In Figure 3 below, the status summary page includes options to save the current status in pdf format, as well as the option to return back to the main screen to facilitate any updates.

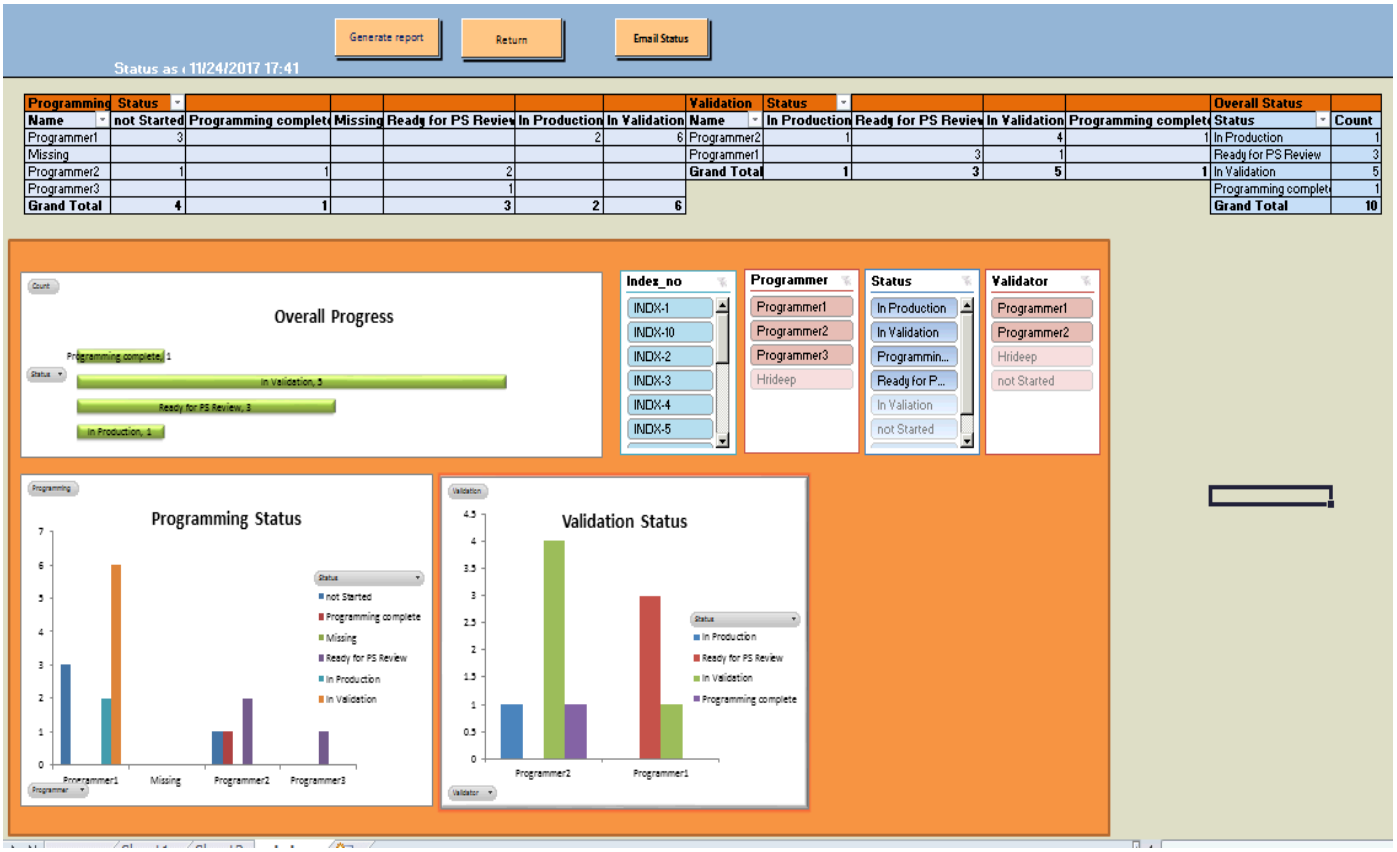

Figure 3: Excel VBA Status Summary Screen

The "**Generate report**" button on the status report page will trigger a prompt box, which will allows users to save the status reports in a pdf format. The VBA code is shown as follows:

```
Private Sub generate_report()
'This  coverts the whole status summary sheet into .pdf file format
Dim Ws As Worksheet
Dim Filename As String, myShell As Object

Filename = Application.InputBox("Enter the pdf file name", Type:=2)

Sheets("status").Range("A4:q100").ExportAsFixedFormat Type:=xlTypePDF, _
    Filename:="Z:\folder1\" & Filename

 MsgBox "Status Report Saved Sucessfully!"
End Sub
```

The "**Return**" button will allow users to return to the original repository front end page to make changes.

```
Private Sub return_tab()
'returns control to the main sheet
ThisWorkbook.Sheets("sheet1").Activate

End Sub
```

CONCLUSION

This dynamic utility will assist users in their ability to achieve a good understanding of the project status while improving the efficiency of the program developmental cycle and programming compliance due to process automation.

This utility is the pilot of the capabilities of the "smart program repository" automation tool, which will be incorporated into another tool being used for tracking projects, Jira. For more details on Jira, please refer to PharmaSUG 2018 Paper LD-02 by Nancy Brucken, "Customized Project Tracking with SAS and Jira."

Refer to Appendix for the SAS® utility macro that creates the shell programs.

REFERENCES

Four Useful VBA Utilities for SAS® Programmers, Available at URL:

https://www.pharmasug.org/proceedings/2013/BB/PharmaSUG-2013-BB11.pdf

Give the Power of SAS® to Excel Users Without Making Them Write SAS Code, Available at URL:

https://www.pharmasug.org/proceedings/2013/TF/PharmaSUG-2013-TF09.pdf

Generating Microsoft Word Macros that Automate the Organization and Maintenance of SAS Tables, Listings and Figures,Available at URL:

http://www.pharmasug.org/2005/CC14.pdf

ACKNOWLEDGMENTS

Thanks to our Director Steve Benjamin for his leadership, constant support, encouragement and valuable assistance in reviewing this paper. Thanks also to our colleague Shelley Murdock for reviewing the paper and providing valuable suggestions.

CONTACT INFORMATION

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

Hrideep Antony
Principal Statistical Programmer, Clinical Division, Syneos Health
Phone: 919-337-1415
E-mail:Hrideep.antony@inventivhealth.com
Web: http:/www.syneoshealth.com

Aman Bahl
Senior Manager, Statistical Programming, Clinical Division, Syneos Health
Phone: 289-313-3014
E-mail:Aman.Bahl@syneoshealth.com
Web: http:/www.syneoshealth.com

APPENDIX

```
%macro createshell;

/*************************************************************************************************************************************************/
/*Step1: Read the input file and  assign macro parameters to program names based on the number of records with "not started" status"*/
/*************************************************************************************************************************************************/
data indata;
set xdb.sheet1(rename=(index_no=index_no_)where =(sr_no >. and upcase(Status) =upcase( 'not Started') and Program_name ne ''));
index_no=_n_;
 call symput("index_no",compress(_n_));
 if index(program_name,'.')>0 then  s_program=substr(program_name,1,index(program_name,'.')-1);
   else s_program=program_name;
     outputlocation=CATX('/',Folder_location,  program_name);
run;

options mprint mlogic;

%macro headerpopulate;
/*************************************************************************************************************************************************/
/*Step2: for each of the programs that needs to be created create the shell using the parameters defined in the excel */
/*************************************************************************************************************************************************/
%do i= 1 %to &index_no;

 proc transpose data=indata out=indata_transp;
 var Programmer --outputlocation;
  by index_no;
 run;
/*************************************************************************************************************************************************/
/*Assign macro parameters to each of the headers parameters based on the input file*/
/*************************************************************************************************************************************************/
data symput_data_&i;
set indata_transp;
  where index_no=&i;
 call symput( "Macro_"!!compress(_name_),  trim(col1) );
 run;
 /*************************************************************************************************************************************************/
/*Step 3: Using the filename statement read all the exisiting files under that directory */
 /*************************************************************************************************************************************************/
Filename Filename   pipe "dir  &Macro_Folder_location ";
%let filefound=0;
   Data _file_exist&i  ;
     Infile Filename truncover;
     Input filename $200.;
     Put filename=;
     if filename=:"dir: cannot access" then fileok=0;
      else fileok=1;
/*************************************************************************************************************************************************/
 /*If the folder location does not exisit then write a message */
 /*************************************************************************************************************************************************/
```

```
  %put Message : File &macro_outputlocation already exists;
  %end;
 /*************************************************************************************************************************************************/
 /*Step 4: If the folder location is correct and the file name is approved then proceed to create the shell program */
 /*************************************************************************************************************************************************/
 %if &filefound =0 %then %do;
%let shellpath=%str(/_shell.sas);
/*Reading the shell program and assigning macro to each line*/
data my_header;
  infile  "&shellpath" truncover;
  input org_prgm $200.;
  x=_n_;
  call symput("code_"!!compress(x),org_prgm);
  if index(org_prgm,'*eoh**')>0 then call symput ("counter",_n_);
run;

%let _code_name=test;
%put &code_4 &counter;

 %put ***** &Macro_Folder_location &macro_Folder_location/&macro_Program_name ;
 /*************************************************************************************************************************************************/
 /*Step 5: Output the resolved shell programs with the appropriate names */
 /*************************************************************************************************************************************************/
%macro create_shell;
data resolved_header;
length test $1000.;
 %do k =1 %to &counter;
  resolved_code=%str("&code_&k");
   x=&k;
   output;
 %end;
 run;

data shell_&macro_s_program ;
  merge resolved_header(in=a) my_header;
  by x;
  if a then Shell_program=resolved_code;
   else Shell_program=org_prgm;
 run;


 filename tmpwhen "&macro_outputlocation"; /* Creates a SAS program file */

 data _null;
  set shell_&macro_s_program END=last;
   format org_prgm $1000.;
   file tmpwhen; /* This is the file the PUT statements writes to */
    put  Shell_program;
 run;

 data toprint ;
   Message="Congrats!  File &macro_outputlocation created successfully";
 run;

 proc print data= toprint noobs;
 run;

 %mend;
 %create_shell;
 %end;
 %end;

 %mend headerpopulate;

 %headerpopulate;

 %mend createshell;
```