PharmaSUG 2011 - Paper TT08

Alternative Approaches to Creating Disposition Flow Diagrams

Brian Fairfield-Carter, ICON Clinical Research, Redwood City, CA
Suzanne Humphreys, ICON Clinical Research, Redwood City, CA

ABSTRACT

The description of subject screening, enrollment, randomization, completion and discontinuation is generally referred to as ‘subject disposition’. Subject disposition tends to be reported in tabular format, providing frequency counts in various disposition categories, occasionally supplemented by ‘time-to-event’ graphs for subject discontinuation. However, since disposition basically describes how subjects ‘flow’ though the life of the project, the clearest and most intuitive means of presenting is with a flow diagram. The problem is that there is no SAS procedure especially well-suited for creating flow diagrams, probably because the layout and content of flow diagrams is highly variable and unpredictable. This paper illustrates and discusses the pros and cons of three alternative methods for creating disposition flow diagrams:

- Creating flow diagrams using only the Annotate facility
- Creating flow diagram elements using Annotate, and arranging and linking them in Excel
- Using Windows scripting to transcribe values into a template or mock-up

INTRODUCTION

By examining subject ‘flow’ through a trial, we can gain useful information not only on safety and efficacy, but also on the efficiency and effectiveness of trial design and conduct. For example, rates of discontinuation due to lack of efficacy or due to adverse events provide indirect but valuable efficacy and safety measures, and rates of screen failures can provide useful measures of the effectiveness of recruitment procedures. Such disposition measures are usually given in tabular format (occasionally supplemented with ‘time-to-discontinuation’ curves), but the trend seems to be to move toward a more visual representation, specifically in the form of disposition flow diagrams.

Disposition flow diagrams are usually mocked up in Word, and because (a) flow diagram design tends to be highly variable, and (b) there is no SAS procedure tailored to creating flow diagrams, the complete flow diagram is often created by manually transcribing disposition values from summary tables directly into the mock-up.

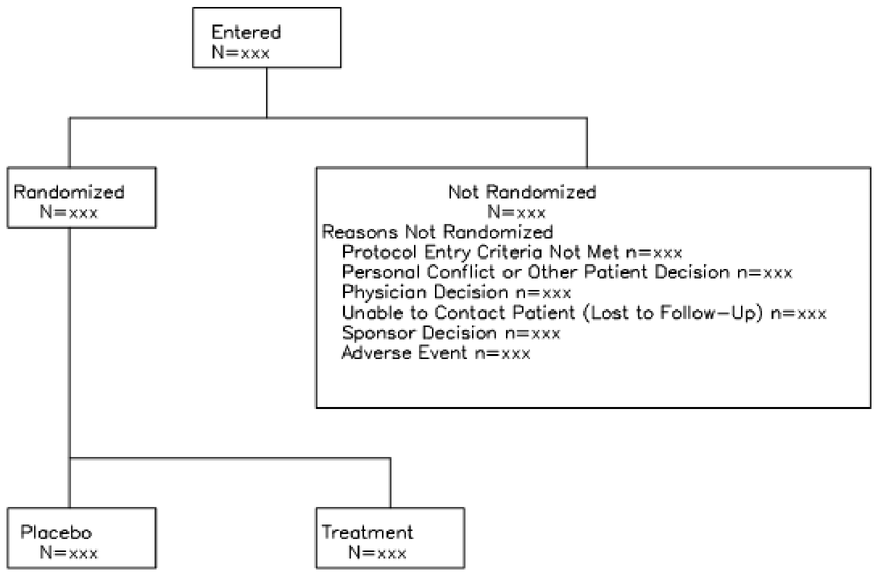

Figure 1: A Simple Disposition Flow-Diagram

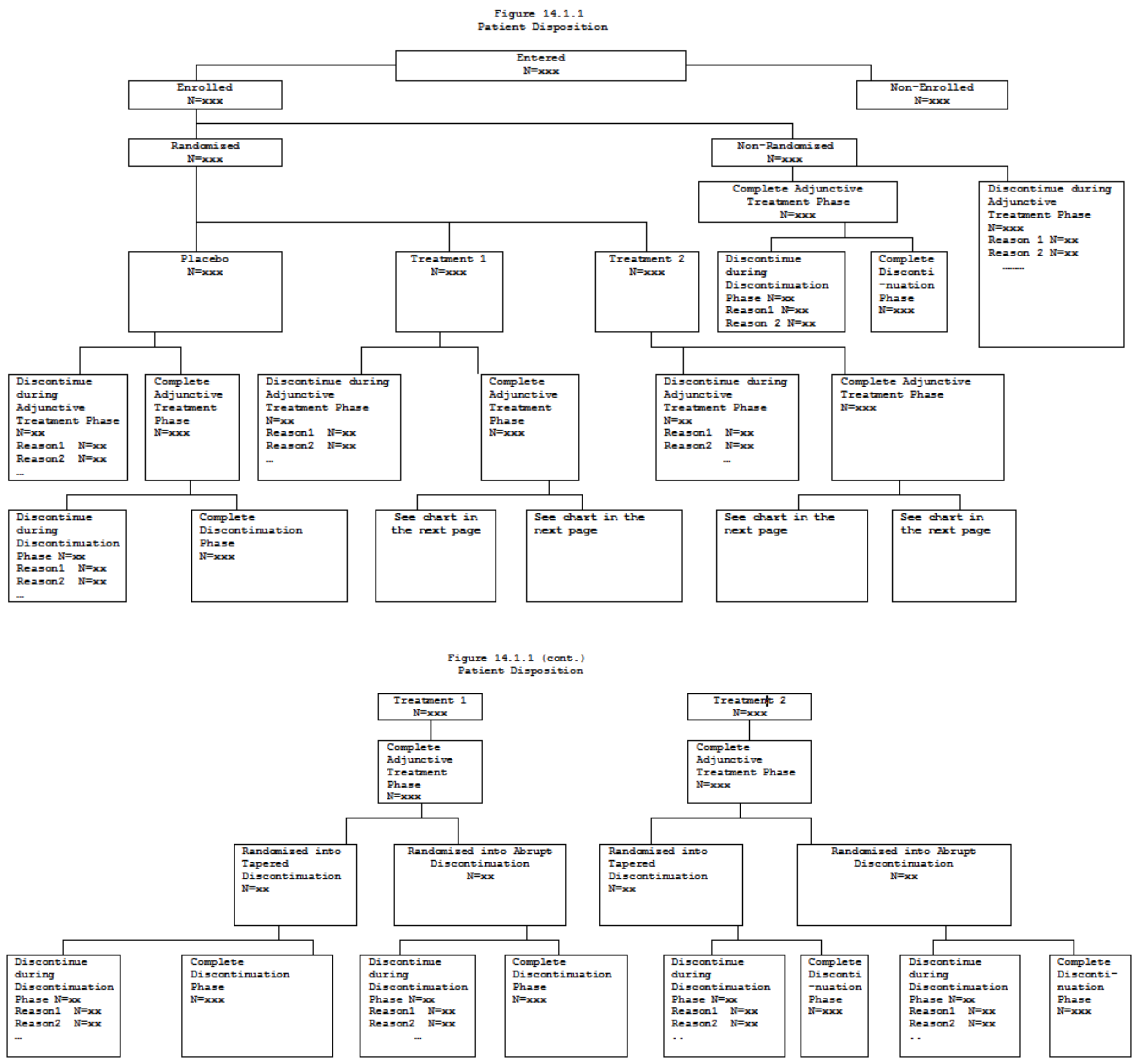
Figure 14.1.1
Patient Disposition
Entered N=xxx
Enrolled N=xxx
Non-Enrolled N=xxx
Randomized N=xxx
Non-Randomized N=xxx
Complete Adjunctive Treatment Phase N=xxx
Discontinue during Adjunctive Treatment Phase N=xxx Reason 1 N=xx Reason 2 N=xx ……
Placebo N=xxx
Treatment 1 N=xxx
Treatment 2 N=xxx
Discontinue during Discontinuation Phase N=xx Reason1 N=xx Reason 2 N=xx
Complete Disconti -nuation Phase N=xxx
Discontinue during Adjunctive Treatment Phase N=xx Reason1 N=xx Reason2 N=xx …
Complete Adjunctive Treatment Phase N=xxx
Discontinue during Adjunctive Treatment Phase N=xx Reason1 N=xx Reason2 N=xx …
Complete Adjunctive Treatment Phase N=xxx
Discontinue during Adjunctive Treatment Phase N=xx Reason1 N=xx Reason2 N=xx …
Complete Adjunctive Treatment Phase N=xxx
Discontinue during Discontinuation Phase N=xx Reason1 N=xx Reason2 N=xx …
Complete Discontinuation Phase N=xxx
See chart in the next page
See chart in the next page
See chart in the next page
See chart in the next page
Figure 14.1.1 (cont.)
Patient Disposition
Treatment 1 N=xxx
Treatment 2 N=xxx
Complete Adjunctive Treatment Phase N=xxx
Complete Adjunctive Treatment Phase N=xxx
Randomized into Tapered Discontinuation N=xx
Randomized into Abrupt Discontinuation N=xx
Randomized into Tapered Discontinuation N=xx
Randomized into Abrupt Discontinuation N=xx
Discontinue during Discontinuation Phase N=xx Reason1 N=xx Reason2 N=xx …
Complete Discontinuation Phase N=xxx
Discontinue during Discontinuation Phase N=xx Reason1 N=xx Reason2 N=xx …
Complete Discontinuation Phase N=xxx
Discontinue during Discontinuation Phase N=xx Reason1 N=xx Reason2 N=xx . .
Complete Disconti -nuation Phase N=xxx
Discontinue during Discontinuation Phase N=xx Reason1 N=xx Reason2 N=xx . .
Complete Disconti- nuation Phase N=xxx

Figure 2: A Complicated, 2-Page Disposition Flow Diagram

Some flow diagrams (i.e. Figure 1) are fairly simple, and don't report many values; others (i.e. Figure 2), as required by complex study design, can be highly complex, span multiple pages, and report large numbers of values. This means that while manual transcription is not really a viable solution, its not immediately obvious what the best approach would be for automating production. On one hand, the entire diagram could be created using the Annotate facility, but while this wouldn't be a huge task for a simple flow diagram, it would require extensive development work for a large and complex flow diagram.

The objective of this paper is to describe the basic technical requirements for creating flow diagrams in Annotate, and discuss the relative merits of this exclusively SAS-based approach compared to two 'hybrid' alternatives: using Annotate to create individual elements (boxes and text) which are then imported into Excel and linked using connecting lines, and using Windows scripting to automate the task of transcribing disposition values calculated in SAS into a mock-up or flow diagram template (Word document).

CREATING FLOW DIAGRAMS USING ANNOTATE

While there is no SAS procedure designed specifically to create flow diagrams, the Annotate facility does provide the ability to produce all the building blocks (rectangles, text, and connecting lines). Annotate consists of a series of drawing functions (i.e. draw, label) and attributes (color, size, etc.), which are rendered via SAS/Graph procedures.

SAS provides a number of Annotate macros (i.e. %rect, %label, %line) that generate function calls, and greatly reduce programming overhead.

The following code fragment illustrates the basics of creating a flow diagram using Annotate (it generates the diagram shown in Figure 1). The building blocks for the diagram are created via the %rect, %label and %line Annotate macros (made available to the session via the '%annomac' call). Positioning of text within each rectangle is calculated dynamically based on rectangle position and size, but for simplicity, rectangle x,y coordinates and size, and line positioning and length are simply hard-coded (rather than attempting to calculate these dynamically). Naturally the 'production' flow diagram would need to replace the 'xxx' place-holders with the actual disposition values.

```
   goptions gunit=cells HSIZE=8 in VSIZE=6 in;
```

Make Annotate macros available to the current session:

```
   %annomac;
```

Create rectangles and text (parameters 'x' and 'y' set the coordinates within the procedure output area (xsys='6', ysys='6') of the top left corner of the rectangle, and parameters 'width' and 'height' set the dimensions of the rectangle; the 'text' parameter receives the text to display in the rectangle, and if there are several lines of text they are separated by a pipe delimiter):

```
   %macro add_element(x=,y=,width=,height=,text=);

     %rect(&x,&y,&x+&width,&y+&height,black,1,0.2);

     %let i=1;
     %do %until (%qscan(&text,&i,'|')=);

       %label(&x+0.5,&y+&height-&i,"%qscan(&text,&i,'|')",BLACK,0,0,1,SIMPLEX,6);
       %let i=%eval(&i+1);

     %end;

   %mend add_element;
```

Create Annotate dataset containing rectangle, label, and line drawing objects:

```
   data final;
     length function color style $8. text $60.;
     retain xsys '6' ysys '6' hsys '6' when 'a' line 1 function 'label';

     %add_element(x=20,y=35,width=12,height=3,text=%str(  Entered |  N=xxx));
     %add_element(x=5, y=27,width=12,height=3,text=%str(Randomized|    N=xxx));
     %add_element(x=30,y=18,width=45,height=12,
     text=%str(            Not Randomized|               N=xxx|Reasons Not Randomized|
     Protocol Entry Criteria Not Met n=xxx|
     Personal Conflict or Other Patient Decision n=xxx|
     Physician Decision n=xxx|
     Unable to Contact Patient (Lost to Follow-Up) n=xxx|
     Sponsor Decision n=xxx|
     Adverse Event n=xxx));
     %add_element(x=5, y=10,width=12,height=3,text=%str( Placebo|    N=xxx));
     %add_element(x=30, y=10,width=12,height=3,text=%str(Treatment|    N=xxx));

     /*** CONNECTING LINES... ***/
     %line(26,35,  26,32.5,BLACK,1,1);
     %line(10,32.5,52,32.5,BLACK,1,1);
     %line(10,32.5,10,30  ,BLACK,1,1);
     %line(52,32.5,52,30  ,BLACK,1,1);
     %line(10,27,  10,13  ,BLACK,1,1);
     %line(10,15.5,36,15.5,BLACK,1,1);
     %line(36,15.5,36,13  ,BLACK,1,1);

   run;
```

Render the Annotate dataset, and write the flow diagram to an .rtf file:

```
   ods rtf file="diagram.rtf";
```

```
    proc gslide annotate=final;
    run;
    quit;

  ods rtf close;
```

This ‘proof of concept’ shows that for small and/or simple flow diagrams, Annotate provides a perfectly feasible solution. Certainly far greater sophistication can be built into calculating the position and size of the various elements, meaning a much more generalized flow-diagram-generating macro can be created (for a good example, refer to Saradha & Veeravel (2008)). However, consider the flow diagram given in Figure 2. This diagram contains a large number of elements, arranged in complicated ways (for example, inconsistent and non-symmetrical ‘tiers’). This means that the number of Annotate statements required would greatly increase, as would the complexity of a macro that attempted to generalize flow diagram layout. This raises the question of whether at some point it becomes preferable to split up the tasks, using SAS for those tasks to which it is well suited (i.e. calculating disposition values) and using something else for tasks that SAS is not well-suited to.

CREATING FLOW DIAGRAMS USING ANNOTATE, EXCEL, AND WINDOWS SCRIPTING

As we can see from the previous example, the Annotate facility can produce individual flow-diagram elements (boxes and text) with very little code; the real complexity, which increases with more complicated flow diagrams, lies in arranging these elements on the page and linking them with connecting lines. The approach presented here splits up these tasks: Annotate is used to create individual elements (each box and the text it contains is stored in a separate graphics output file), and these elements are imported, arranged and linked in Excel. The tasks carried out in Excel can be automated by recording the actions as a VBA macro (which can then be adapted as a ‘stand-alone’ Visual Basic script), which saves the programmer the laborious task of determining coordinates for placement of individual elements and lines.

The first step is to create the individual elements, stored in separate graphics files. Note that goptions HSIZE, VSIZE, HPOS and VPOS are re-set for each element, since according to SAS online documentation, setting HSIZE/VSIZE and HPOS/VPOS “changes the external dimensions of the graphics output area and recalculates the number of rows and columns in order to retain cell size and proportions” (which in turn preserves a consistent text size regardless of the dimensions of the individual flow-diagram element). In the following example, flow diagram elements are stored in a series of .emf files (_1.emf, _2.emf,…, _<n>.emf), which are then imported and arranged in Excel. The code is essentially a slightly reorganized version of the previous example; the only difference is that each element is rendered and output separately, and the graphics output area is reset (via the ‘xmax’ and ‘ymax’ parameters) for each element in order to preserve text size.

```
  %let fileno=0;
  %macro add_element(xmax=,ymax=,x=,y=,width=,height=,text=);

    %let fileno=%eval(&fileno+1);
    filename fileref "_&fileno..emf";

    %*** SET HSIZE/VSIZE AND HPOS/VPOS TO PRESERVE TEXT SIZE;
    goptions device=sasemf
             gsfname=fileref
             gsfmode=replace
             gunit=cells
             HSIZE=&XMAX in
             VSIZE=&YMAX in
             HPOS=&width
             VPOS=&height;

    data anno;
      length function color style $8. text $60.;
      retain xsys '6' ysys '6' hsys '6' when 'a' line 1 function 'label';
      %rect(&x,&y,&x+&width,&y+&height,black,1,0.2);

      %let i=1;
      %do %until (%qscan(&text,&i,'|')=);
        %label(&x+0.5,&y+&height-&i,"%qscan(&text,&i,'|')",BLACK,0,0,1,SIMPLEX,6);
        %let i=%eval(&i+1);
      %end;
    run;
    proc gslide annotate=anno gout=fileref;
    run;
    quit;

  %mend add_element;


  %add_element(xmax=1,ymax=0.5,x=20,y=35,width=12,height=3,text=%str(  Entered |
  N=xxx));
  ...(etc.)...
```

The flow diagram elements, stored as individual graphics files, can now be imported (using the 'Insert/Picture' drop-down menu), arranged and connected in Excel (figure 3). Connecting lines are simply created using the drawing tool.

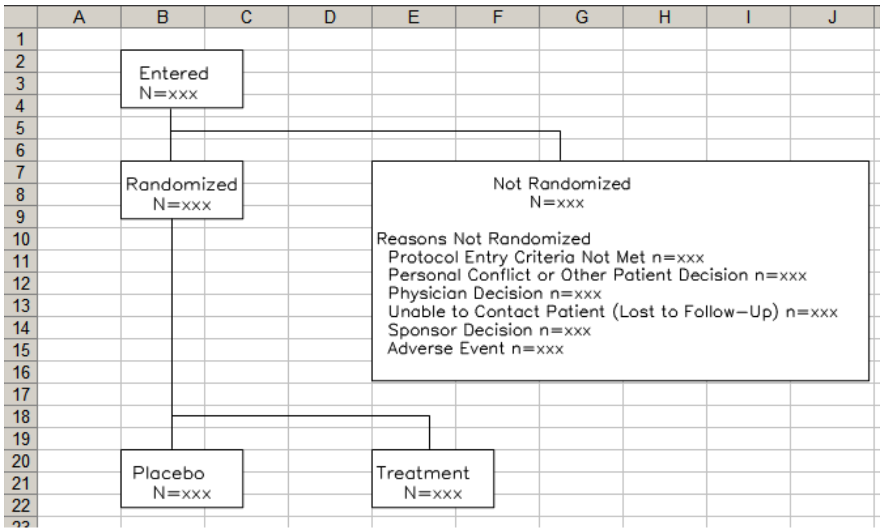

Figure 3: Flow-Diagram Elements Imported and Arranged in Excel

This step can be automated, by recording a macro while arranging the flow diagram and (optionally) adapting the generated VBA code as a stand-alone VBScript (see Hunt *et al* (2005)) for details). This is useful since the flow diagram will probably need to be recreated each time the source data are updated. The following is a VBScript used to created the flow diagram shown in figure 3; note that the easiest way to capture cell references ('Range') and line coordinates ('AddLine') is by first arranging the flow diagram by hand while recording a macro, and then retrieve the references from the macro when creating the stand-alone script.

```
  dim objXLS, wshell, fso, GetBook
```

Determine the current working directory:

```
  set fso= CreateObject("Scripting.FileSystemObject")
  Set wshell = CreateObject("WScript.Shell")
  set f = fso.GetFolder(wshell.currentdirectory)
```

Launch Excel and add a blank workbook:

```
  Set objXLS = WScript.CreateObject("Excel.Application.11")
  objXLS.visible=TRUE

  Set GetBook = objXLS.Workbooks.Add
```

Insert SAS-generated graphics (in this case .emf) files into target cells:

```
  with objXLS
    .Range("B2").Select
    .ActiveSheet.Pictures.Insert(f & "\_1.emf").Select
    .Range("B7").Select
    .ActiveSheet.Pictures.Insert(f & "\_2.emf").Select
    .Range("E7").Select
    .ActiveSheet.Pictures.Insert(f & "\_3.emf").Select
    .Range("B20").Select
    .ActiveSheet.Pictures.Insert(f & "\_4.emf").Select
    .Range("E20").Select
    .ActiveSheet.Pictures.Insert(f & "\_5.emf").Select
```

Draw connecting lines to complete the flow diagram:

```
    .ActiveSheet.Shapes.AddLine(76.5, 46.5, 76.5, 76.5).Select
    .ActiveSheet.Shapes.AddLine(76.5, 59.25, 300, 59.25).Select
    .ActiveSheet.Shapes.AddLine(300, 59.25, 300, 75.75).Select
    .ActiveSheet.Shapes.AddLine(77.25, 109.5, 77.25, 241.5).Select
    .ActiveSheet.Shapes.AddLine(77.25, 222.75, 225, 222.75).Select
    .ActiveSheet.Shapes.AddLine(225, 222.75, 225, 241.5).Select
  end with
```

Save the completed flow diagram, and close Excel:

```
  objXLS.ActiveWorkbook.SaveAs f & "\flow_chart.xls"
  objXLS.application.quit
```

This approach is appealing since the association between disposition value and label is ‘fixed’ in the SAS/Graph output, while leaving the tasks that SAS is not particularly amenable to (namely the arrangement and connecting of flow diagram elements) to an environment better suited to the task; at the same time, the task of arranging and connecting flow diagram elements can be automated in anticipation of future updates to the disposition values and to the content of individual elements.

CREATING FLOW DIAGRAMS BY TRANSCRIBING VALUES INTO A TEMPLATE

The project statistician usually puts considerable effort into mocking up the disposition flow diagram in Word, and in some cases the ‘production’ figure is simply the mock-up with the actual disposition values manually transcribed into it. Windows scripting technologies are ideally suited to automating these sorts of tasks. In short, all that is required is to set up a series of ‘search/replace’ commands that will replace a ‘token’ or place-holder in the mockup with the actual derived value. The tremendous advantage of this approach is that it ‘leverages’ the set-up work done by statistician, making use of the formatting capability of Word (meaning it avoids the extensive coding associated with creating and positioning the flow-chart elements). The draw-back is that as with the preceding example it turns the production of the flow diagram into a two-step process (even more so in this case, since the disposition values are derived in SAS, and inserted into the mock-up as a post-processing step, meaning the association between value and label isn’t established until this post-processing step).

To implement this approach, all that is required is for each place in the mockup where ‘N=xxx’ is displayed, to replace the ‘xxx’ with a unique identifier (for example, ‘N=VAR1’).

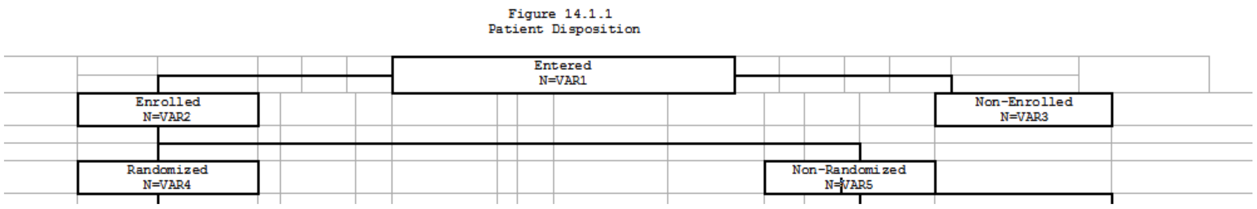

Figure 4: Flow-Diagram Mock-up With ‘Find/Replace’ Place-Holders

In SAS, when disposition frequency counts are calculated, simply write out a text file containing a list of these unique identifiers and their associated disposition values, separated by a delimiter, for example:

```
  data _null_;
    set disposition;
    file replacevalue.txt;
    put "VAR"||put(_n_,best.)||"|"||put(count,best.);
  run;
```

This produces:

```
VAR1|200
VAR2|180
VAR3|20
VAR4|150
VAR5|30
---(etc.)---
```

Once this is done, a script (in the example below, a VBScript, simply a text file saved with a .vbs extension) can be run to capture each identifier/value pair and pass them to a find/replace command, which executes the substitution in Word. The '.vbs' file extension associates the script file with the Windows Scripting Host (for more detail, refer to Hunt *et al* (2005)), which interprets the script and executes the commands. The script does the following:

1. Determines the pathname of the current directory (the assumption being that the script file and the flow diagram mock-up are stored in the same directory)
2. Opens the text file containing the list of place-holders and associated disposition values (the file is assumed to be named "replacefile.txt" and stored in the same location as the script file and mock-up)
3. Launches Word and opens the flow diagram mockup (assumed to be named "mockup.doc")
4. Iterates through each pair of place-hold/disposition values in the "replacefile.txt" text file, searches for the place-holder in the mock-up, and replaces it with the associated disposition value
5. Saves the completed flow diagram as "flow_chart.doc" and closes Word

The complete script is as follows:

```
dim objwd, wshell, fso, replacefile, string
```

Determine the current working directory:

```
set fso= CreateObject("Scripting.FileSystemObject")
Set wshell = CreateObject("WScript.Shell")
set f = fso.GetFolder(wshell.currentdirectory)
```

Open the SAS-generated text file containing the identifier/value pairs:

```
Set File = FSO.GetFile(f & "\replacefile.txt")
Set textin = File.OpenAsTextStream
```

Launch Word, and open the mock-up/template:

```
set objwd = wscript.createobject("word.application")
objwd.visible=TRUE

objWD.Documents.Open(f & "\mockup.doc")
```

For each identifier/value pair, execute a 'find/replace':

```
  Do While Not textin.AtEndOfStream
    string = textin.ReadLine & NewLine

    FindString=Split(string,"|")(0)
    ReplaceString=Split(string,"|")(1)

    objwd.Selection.Find.execute FindString
    objwd.selection.typetext ReplaceString

  Loop

textin.close
```

Save the completed flow diagram and close Word:

```
objWD.ActiveDocument.SaveAs(f & "\flow_chart.doc")

WScript.Sleep 1000
objwd.Documents.Close

objwd.Application.Quit
```

An excerpt of the completed flow diagram is shown in Figure 5:

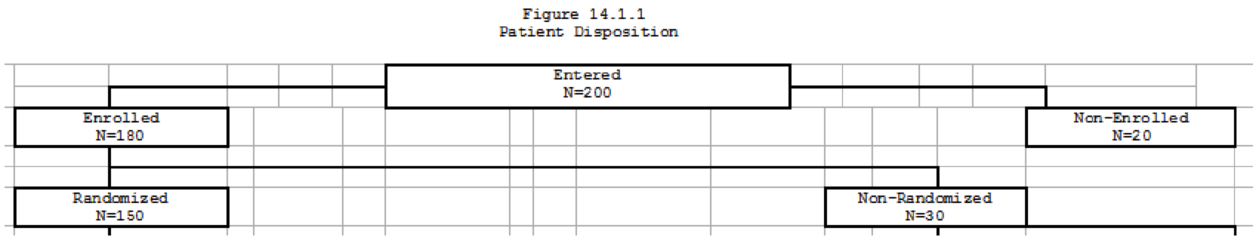

Figure 5: Completed Flow-Diagram

This approach presents an additional advantage where specific text-formatting and style requirements exist: the attributes set up in the mock-up or template are retained through the find/replace. For example, consider if particular values needed to be highlighted via text color, underlining, italicizing or other attributes:

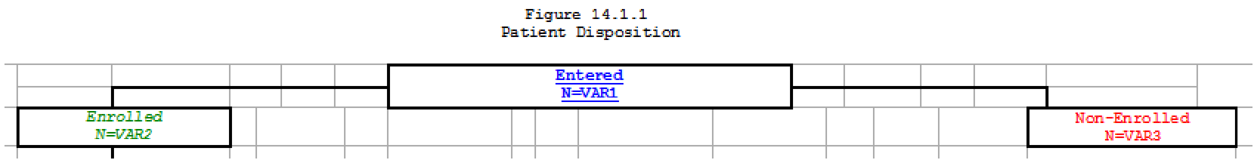

Figure 6: Mockup With Specific Text Attributes

After the script has been run, the substituted values retain the attributes set in the template:

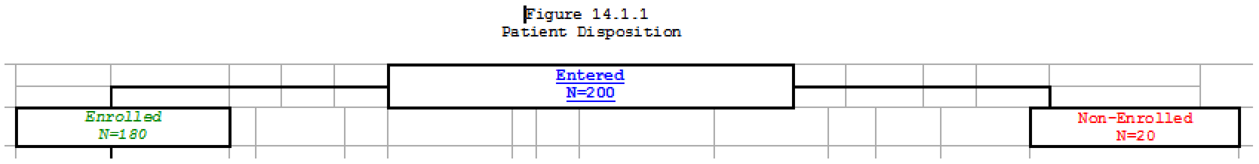

Figure 7: Completed Flow-Diagram Retaining Text Attributes

SUMMARY: PROS AND CONS OF ALTERNATIVE METHODS

As with most programming tasks, the approach selected must take into account a number of inherent tradeoffs. An entirely SAS-based approach may be more 'seamless' and more easily fall within normal QC/validation practices, but may be less flexible and more cumbersome, and require a greater investment of time and effort. The alternate methods proposed may be more flexible and require less time to implement, but introduce 'weak links' that must be covered by more labor-intensive QC practices. Table 1 summarizes the key tradeoffs associated with each method.

Method	Pros	Cons
1. Annotate	Entirely in SAS; does not require knowledge of scripting Platform-independent	May involve considerable programming overhead, which increases with increasing flow-diagram complexity Limited control over text attributes
2. Annotate, VBS, Excel	Elements created in SAS (association between disposition values & labels is 'fixed' in SAS) Lower programming overhead than with method 1, and little increase associated with increasing flow-diagram complexity	Limited control over text attributes 2-step process Not platform-independent (requires Windows) Requires some scripting knowledge
3. Transcription, VBS, Word	Very low programming overhead; essentially no increase associated with increasing flow-diagram complexity Unlimited control over text attributes (since text attributes are assigned manually in the template/mock-up)	2-step process Not platform-independent Requires some scripting knowledge Association between disposition values & labels takes place external to SAS, which may require atypical QC practices.

Table 1: Tradeoffs associated with each method

CONCLUSION

While disposition flow diagrams can provide valuable insight on safety, efficacy, and trial design, there is no obvious 'best practice' for creating them. SAS-centric approaches using Annotate are cumbersome and time-intensive for large and/or complicated flow diagrams; alternate approaches, while potentially more flexible, may require knowledge of Windows scripting and may not fit nicely with typical QC & validation practices. Hopefully the examples presented here will provide a basis for determining, in light of the various trade-offs, the best approach for a given instance.

REFERENCES

Stephen Hunt, Tracy Sherman and Brian Fairfield-Carter. *An Introduction to SAS® Applications of the Windows Scripting Host*. Proceedings of the 2005 SAS Users Group International (SUGI) Conference.

Priya Saradha & Gurubaran Veeravel. *Creating Flowcharts Using the Annotate Facility*. Proceedings of the 2008 SAS Global Forum Conference.

ACKNOWLEDGMENTS

We would like to thank ICON Clinical Research for consistently encouraging and supporting conference participation, and all our friends at ICON for their great ideas, enthusiasm and support, in particular Syamala Schoemperlen and Jackie Lane.

CONTACT INFORMATION

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

Name: Brian Fairfield-Carter, Suzanne Humphreys
Enterprise: ICON Clinical Research, Inc.
City, State ZIP: Redwood City, CA
E-mail: fairfieldcarterbrian@gmail.com