Paper DV08

A Visual Tool to Aid Traceability

Jesse Pratt, PPD, Williamsburg, OH, USA
Cleopatra DeLeon, PPD, Austin, TX, USA

ABSTRACT

Traceability is a critical component of any clinical research study. This paper discusses the use of interactive flow charts generated in SAS to trace outputs and analysis data sets to their various sources through the course of a study. Chart layouts are designed prior to any programming taking place, followed by the creation of a positioning data set and then generation of the chart itself using the Graph Template Language. The addition of HTML functionality results in a user-friendly, efficient, and interactive way to navigate across data sets and outputs. Illustrative examples highlight each step.

INTRODUCTION

Data traceability is a required virtue in clinical trials where the complete path of distinct values is detailed from the point of capture to final analysis. Knowing where the data came from and where it is ultimately being reported is important not just for regulatory reviewers, but also for sponsors and study team members who are involved in a study's development. Without data traceability, chasing down and explaining data anomalies would be difficult to accomplish.

The approach discussed here was inspired by online genealogy charts, modelling the HTML browsing of parent-child relationships and family groups. This traceability documentation and visualization process is two-fold – first, build flow charts using SAS GTL to illustrate relationships between datasets and outputs, and, finally, enhance user experience with interactivity using HTML hyperlinks. Employing an HTML dashboard, the descendance of a dataset or output is easily traced in a format familiar to many end-users.

METHODOLOGY

In order to successfully build this tool, the following steps must be performed:

1. Design the visual layout
2. Create data file containing text and positioning data
3. Gather necessary outputs, such as tables, listings, figures
4. Program the flow chart(s)
5. Add HTML functionality

DESIGN

Much like outlining a paper or essay prior to writing, sketching a design of the visual layout of the tool will be of great assistance along the way. Consider the following sketch vs. the final flow chart for a specific set of outputs:

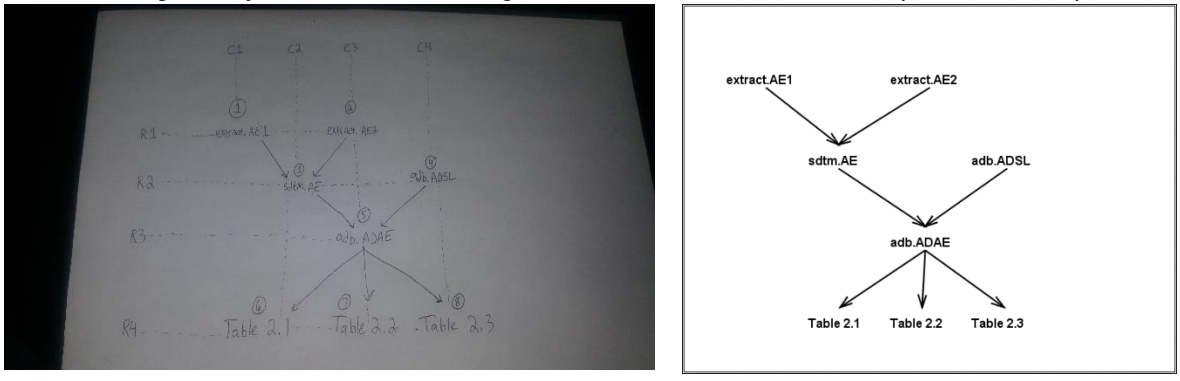

Figure 1: Initial sketch compared to output image

In this sketch, we define row and column numbers, individual numbers for each item in the flow chart, and show the arrows denoting which elements feed into the others. Details will be discussed in subsequent sections.

POSITION AND TEXT DATA

From our sketch, we can populate a data set needed to define positions and fill text boxes based on rows, columns,

the circled numbers, and arrows. The relationship between our sketch and this data set will be discussed in the next section.

GATHER OUTPUTS

Any outputs we wish to link to using HTML functionality need to be generated and stored in the appropriate folder prior to programming the tool itself. Having a standard folder structure facilitates this in an efficient manner, even one as simple as Figure 2.

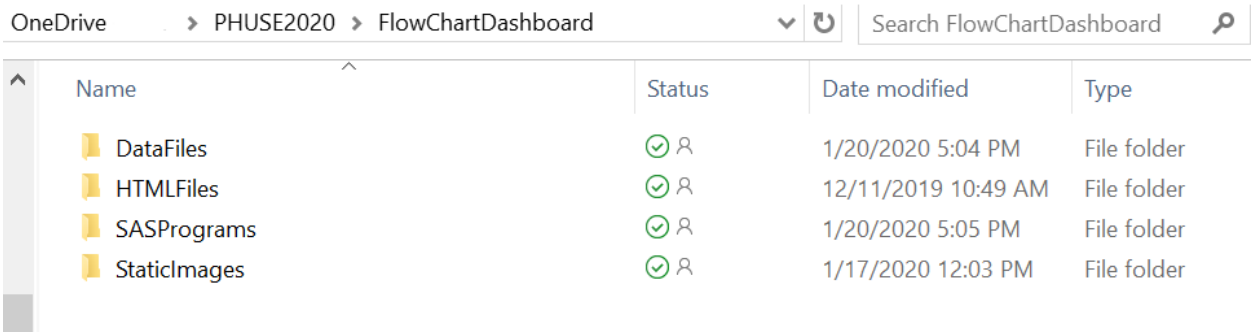

Figure 2: Sample folder structure

OVERVIEW OF SAS PROGRAM

The following list outlines the steps needed to program the flow chart itself in SAS. Details will be provided in the next section.

- Import position and text data set
- From the above data set, find and calculate the maximum row and column numbers, then calculate each row and column as a percentage of the respective maximums
- Create a data set containing coordinates representing a boundary of the graph space; set this with the calculated row and column percentages
- Use GTL to output positioning data for the arrows and place markers on a mockup graph; rerun iteratively until desired positioning obtained
- Merge previous positioning data to the main plotting data set
- Create the coordinates needed to draw the arrows of the flow chart
- Plot the final flow chart

HTML FUNCTIONALITY

Using HTML as the destination for the tool provides the means for interactivity and efficiency for the user. For example, links to the actual outputs can be placed in the respective text boxes within the flow chart, or complex flow charts can be made more efficient with the use of links. Specifics will be provided in the next section.

ILLUSTRATIVE EXAMPLE - ADAE

Details from the prior section will be given by providing an illustrative example, a flow chart for an ADAE analysis data set with associated outputs. Note that all data shown are fabricated and only for illustrative purposes. Let's consider the flow chart we wish to create:

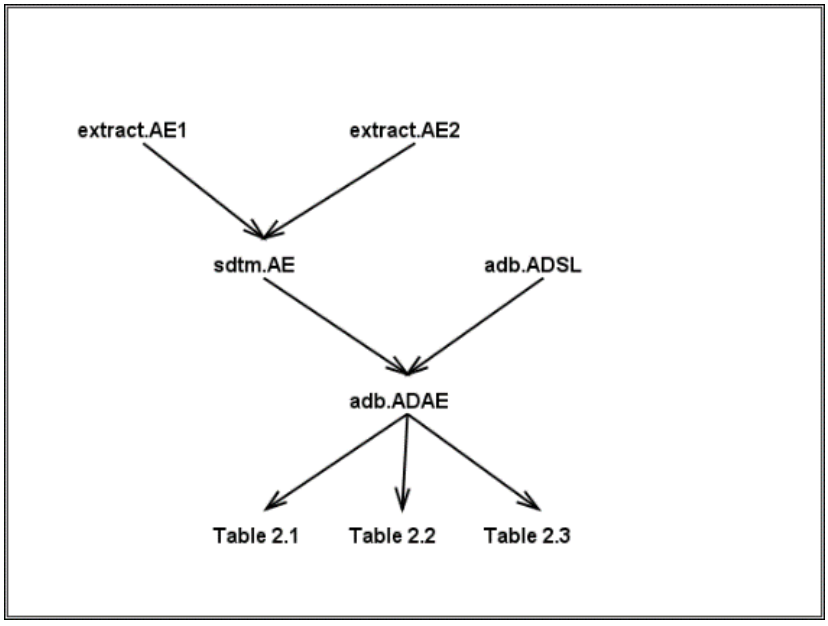

Figure 3: ADAE flow chart

In this chart, we have our adb.ADAE data set creating outputs Table 2.1, 2.2, 2.3. Data sets sdtm.AE and adb.ADSL are both used to create adb.ADAE, and extract.AE1 and extract.AE2 data sets are used to create sdtm.AE. Note that we included libraries EXTRACT, SDTM, and ADB along with the data set names. Analysis data set adb.ADSL will obviously have data sets feeding into it, and documentation of this will be handled toward the end of this example when we discuss HTML functionality.

DESIGN

When designing a flow chart, creating a draft sketch as the first step serves as a helpful outline to refer to throughout this process.

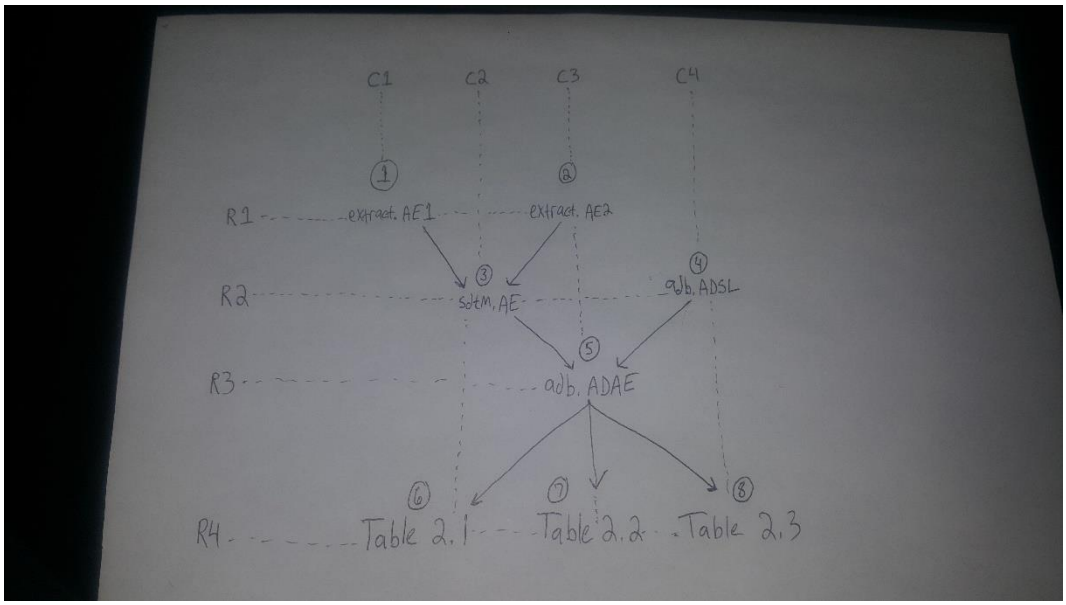

Figure 4: Initial Sketch of ADAE flow chart

Let's discuss some details of this sketch:

- Explicitly write the text as it should appear, along with the appropriate arrows
- The circled numbers represent IDs for each text box, which will assist when it comes time to program
- Arrange the chart into a grid-like structure, so that each text box can be assigned a unique row and column number
- Column 1 contains extract.AE1 (1)

- Column 2 contains sdtm.AE (3) and Table 2.1 (6)
- Column 3 contains extract.AE2 (2), adb.ADAE (5), and Table 2.2 (7)
- Column 4 contains adb.ADSL (4) and Table 2.3 (8)
- Row 1 contains extract.AE1 (1) and extract.AE2 (2)
- Row 2 contains sdtm.AE (3) and adb.ADSL (4)
- Row 3 contains adb.ADAE (5)
- Row 4 contains Table 2.1 (6), Table 2.2 (7), and Table 2.3 (8)

Using this information, we can now build our text/positioning data set.

POSITION AND TEXT DATA

Consider the following data file:

	A	B	C	D	E	F
1	BOXID	ROW	COL	ROUGHTEXT	FROMID	TOID
2	1	1	1	extract.AE1	1	3
3	2	1	3	extract.AE2	2	3
4	3	2	2	sdtm.AE	3	5
5	4	2	4	adb.ADSL	4	5
6	5	3	3	adb.ADAE	5	6
7	6	4	2	Table 2.1	5	7
8	7	4	3	Table 2.2	5	8
9	8	4	4	Table 2.3		

Figure 5: Position and text data set

- The variable BOXID represents the ID number for each text box, represented by the circled numbers in our sketch
- The ROW and COL variables represent the row and column numbers, respectively, of each text box
- ROUGHTEXT is the verbiage to be displayed in each text box. While not applicable in this example, sometimes the inclusion of a split character is necessary.
- The variables ROW, COL, and ROUGHTEXT all correspond to the value of BOXID in the same line; this does not hold true for the variables FROMID and TOID.
- The variable FROMID denotes the starting position of each arrow in terms of the text ID number and the variable TOID denotes the end position of the corresponding arrow. For example, we have an arrow pointing from extract.AE1 (corresponding to FROMID=1) to sdtm.AE (corresponding to TOID=3).

GATHER OUTPUTS

In Figure 2, we see a sample folder structure to be used for this tool. Folders for SAS programs and input data are created. In the StaticImages folder, we store all image files to be used in our tool:

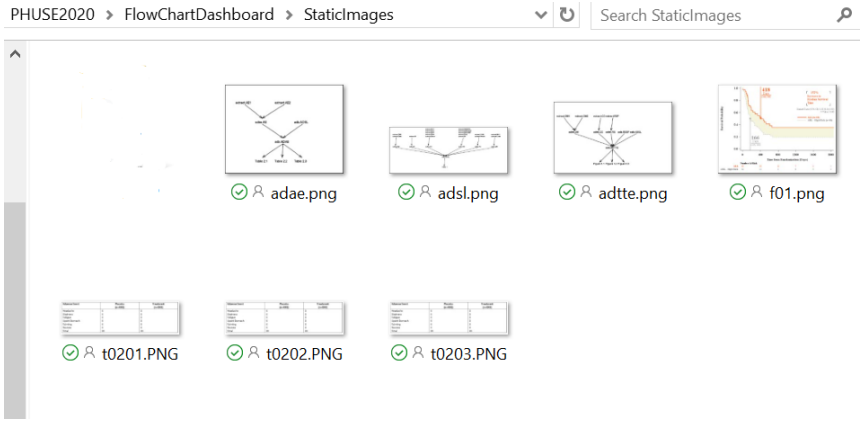

Figure 6: Content of Image folder

Likewise, we have a similar folder containing HTML files, details of which will be discussed in later sections.

SAS PROGRAM

In this section, we will outline the programming steps necessary to build the flow chart for our ADAE data set. Key

programming syntax will be provided, along with data set snapshots, however many explicit coding details will be left to the reader. Our first step will be to import the positioning/text data set into SAS:

BOXID	ROW	COL	ROUGHTEXT	FROMID	TOID
1	1	1	extract.AE1	1	3
2	1	3	extract.AE2	2	3
3	2	2	sdtm.AE	3	5
4	2	4	adb.ADSL	4	5
5	3	3	adb.ADAE	5	6
6	4	2	Table 2.1	5	7
7	4	3	Table 2.2	5	8
8	4	4	Table 2.3	.	.

Figure 7: Initial import

Next, we determine the maximum values of the variables ROW and COL, then compute x- and y-coordinates representing position of each in terms of percentage of the maximum value. The numerators are the respective values of ROW and COL; we wish for each of these positioning coordinates to be less than 100, thus we add 1 to the maximum values to obtain our denominators. In this example, for BOXID=1, our numerator for rows is 1 and denominator for rows is 4+1=5 and so our x-coordinate is 100*(1/5) = 20. Similar calculations give us a y-coordinate of 20. The following data set shows this calculation for each observation:

BOXID	ROW	COL	ROUGHTEXT	FROMID	TOID	roughx	roughy
1	1	1	extract.AE1	1	3	20	20
2	1	3	extract.AE2	2	3	60	20
3	2	2	sdtm.AE	3	5	40	40
4	2	4	adb.ADSL	4	5	80	40
5	3	3	adb.ADAE	5	6	60	60
6	4	2	Table 2.1	5	7	40	80
7	4	3	Table 2.2	5	8	60	80
8	4	4	Table 2.3	.	.	80	80

Figure 8: Calculated positioning coordinates

Next, we begin an iterative process of building a “canvas” to further refine positioning. We will plot these points to create boundaries, using a scatter plot with white marker color. Add these “canvas” variables to the prior data set to obtain:

BOXID	ROW	COL	ROUGHTEXT	FROMID	TOID	roughx	roughy	dummyx	dummyy
1	1	1	extract.AE1	1	3	20	20	.	.
2	1	3	extract.AE2	2	3	60	20	.	.
3	2	2	sdtm.AE	3	5	40	40	.	.
4	2	4	adb.ADSL	4	5	80	40	.	.
5	3	3	adb.ADAE	5	6	60	60	.	.
6	4	2	Table 2.1	5	7	40	80	.	.
7	4	3	Table 2.2	5	8	60	80	.	.
8	4	4	Table 2.3	.	.	80	80	.	.
.	.	.		.	.	.	.	0	0
.	.	.		.	.	.	.	100	100

Figure 9: Addition of canvas variables

Using the ODS GRAPHICS statement, we next set our image to the desired dimensions, in our case a width of 640 pixels and height of 480 pixels:

```
ods graphics / width=640px height=480px;
```

Using the SAS Graph Template Language (GTL), we generate a prototype of the desired graph, minus the arrows:

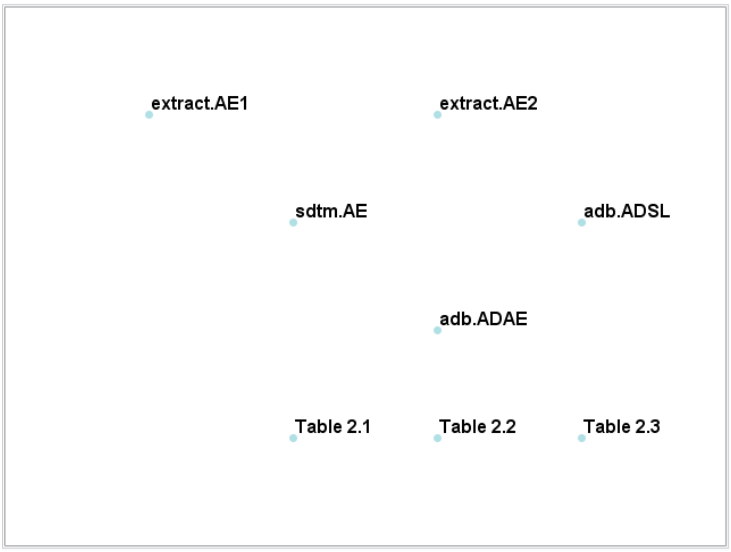

Figure 10: First run plotting text

If we would like to have our text boxes more centered, we can adjust the values of our DUMMYX and DUMMYY canvasing variables to the following:

dummyx	dummyy
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
10	0
130	90

Figure 11: Adjusted canvas values

Using these new values, we have the following update to our figure:

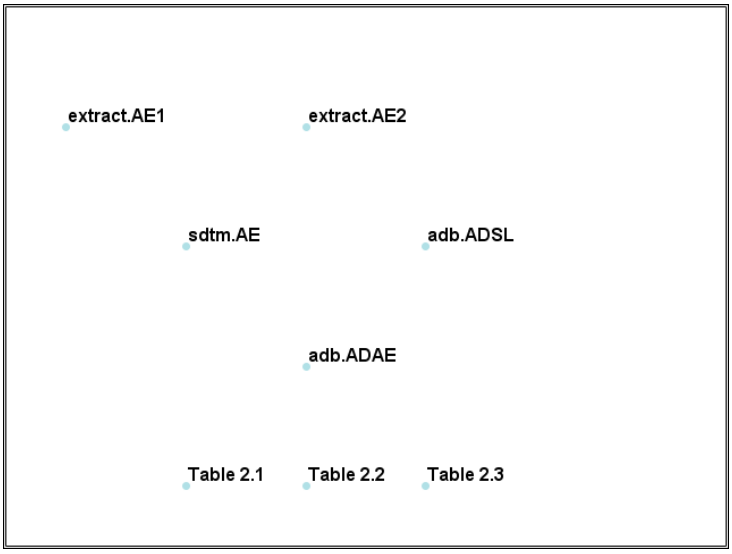

Figure 12: Centered text boxes

The blue markers are present to help us once it comes time to draw the arrows and will be removed prior to finalizing the figure. In the TEXTPLOT statement of the GTL code used to generate this plot, we use the OUTFILE= and OUTID= options to output a data set to provide us with the height and width of each text box. This will be valuable information when creating the arrows.

boxId	DATAWIDTH	DATAHEIGHT
1	17.950338688	5.1125889866
2	17.950338688	5.1125889866
3	13.603892776	5.1125889866
4	16.060579408	5.1125889866
5	16.249555192	5.1125889866
6	14.359795912	5.1125889866
7	14.359795912	5.1125889866
8	14.359795912	5.1125889866

Figure 13: Height and Width data for each text box

Using these data, we merge onto our prior plotting data set and calculate some additional variables to find the centered top and bottom coordinates for each text box:

roughx	roughy	dummyx	dummyy	DATAWIDTH	DATAHEIGHT	BOTTOMX	BOTTOMY	TOPX	TOPY
.	.	10	0	.	.	.	.	.	.
.	.	130	90	.	.	.	.	.	.
20	20	.	.	17.950338688	5.1125889866	28.975169344	20	28.975169344	14.887411013
60	20	.	.	17.950338688	5.1125889866	68.975169344	20	68.975169344	14.887411013
40	40	.	.	13.603892776	5.1125889866	46.801946388	40	46.801946388	34.887411013
80	40	.	.	16.060579408	5.1125889866	88.030289704	40	88.030289704	34.887411013
60	60	.	.	16.249555192	5.1125889866	68.124777596	60	68.124777596	54.887411013
40	80	.	.	14.359795912	5.1125889866	47.179897956	80	47.179897956	74.887411013
60	80	.	.	14.359795912	5.1125889866	67.179897956	80	67.179897956	74.887411013
80	80	.	.	14.359795912	5.1125889866	87.179897956	80	87.179897956	74.887411013

Figure 14: Centered coordinates for the top and bottom of text boxes, merged with other position data

Calculations are as follows:

- BOTTOMX=ROUGHX+0.5*DATAWIDTH
- BOTTOMY=ROUGHY
- TOPX=ROUGHX+0.5*DATAWIDTH
- TOPY=ROUGHY-DATAHEIGHT

From here, this gives us enough to plot the arrows using multiple DRAWARROW annotation statements within GTL, using the FROMID and TOID variables in conjunction with the TOP and BOTTOM coordinates. Remove the blue dots from the prior plot and we arrive at our final flow chart given in Figure 3.

INTERACTIVITY USING HTML

The use of HTML turns this tool from a static image into an interactive way to add detail and improve understanding of data flow. This is done by first creating HTML files from each static image, and then inserting links between files. Let's begin with an image file of a table, and let's create an HTML file from it and place in the appropriate folder. The following HTML code will do this:

```
<html>
   <head>
      <title> Table 2.1 </title>
   </head>

   <body>
      <a href="C:\FlowChartDashboard\StaticImages\t0201.htm">
         <img src="C:\FlowChartDashboard\StaticImages\t0201.png" style="width:640px;
                  height:320px;">
      </a>
   </body>
</html>
```

Subsequent HTML files can be created from images we wish to include in the tool. Next, we turn our attention to the ADAE flow chart. The following code will both create and add hyperlinks to specified areas of the HTML version of our ADAE flow chart:

```
<html>
   <head>
      <title> ADAE </title>
   </head>

   <body>
      <img src="C:\FlowChartDashboard\StaticImages\adae.png" style="width:640px;
            height:480px;" usemap="#adlinks">
      <map name="adlinks">
          <area href="C:\FlowChartDashboard\HTMLFiles\t0201.htm"
            coords="158,421,158,404,232,404,232,421" shape="poly">
          <area href="C:\FlowChartDashboard\HTMLFiles\adsl.htm"
            coords="369,208,369,196,450,196,450,208" shape="poly">
      </map>
  </body>
</html>
```

A few notes on the above code:

- As previously, the `<img>` tag generates an HTML file from the image
- Inside the `<map>` tag, we have two `<area>` tags responsible for creating links to existing HTML files
- The coordinates inside the `<area>` tags define the region in which the links exist

Figures 15 and 16 illustrate the results of this code.

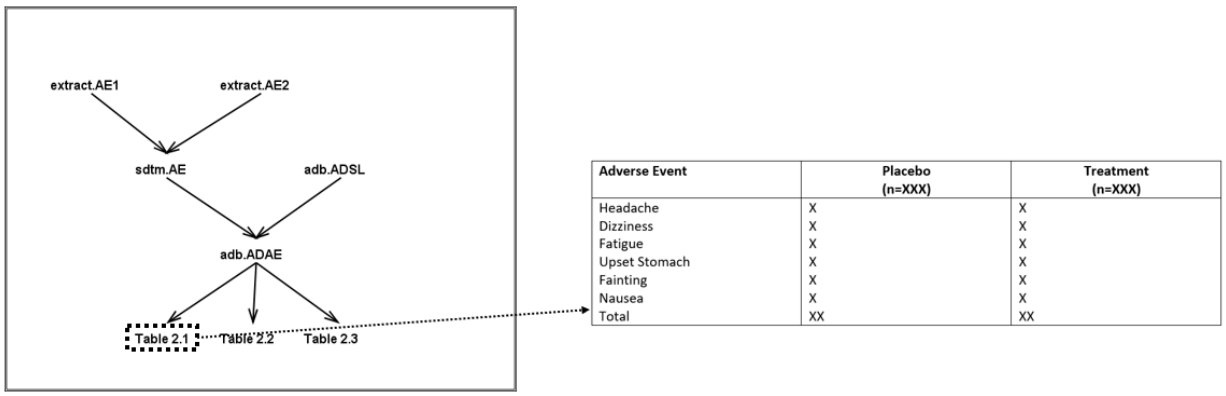

Figure 15: HTML link to table output

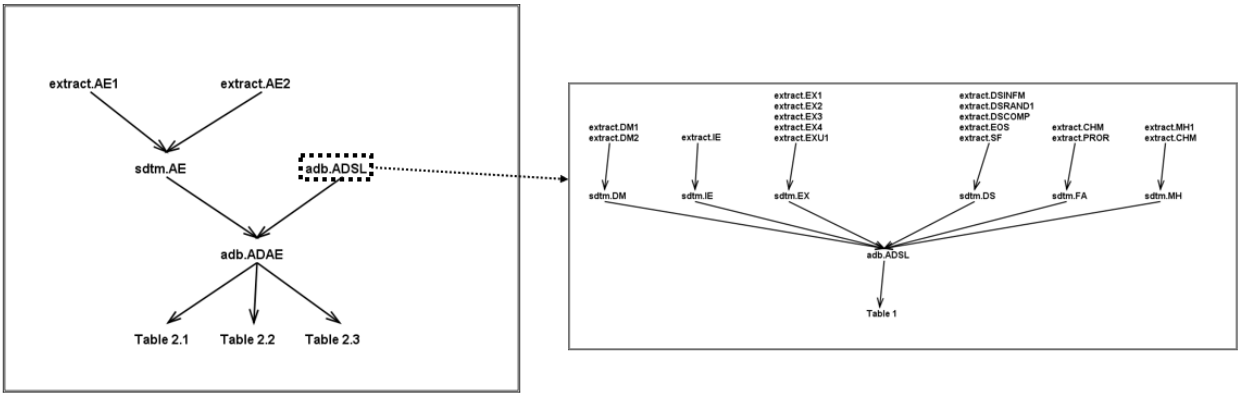

Figure 16: HTML link to another flow chart to preserve space

CONCLUSION

Our method to document and visualize data traceability for a clinical trial using SAS and HTML hyperlinks does not require any specialized software or knowledge beyond a solid understand of SAS graphics and rudimentary knowledge of HTML. However, we acknowledge some limitations in this initial concept. There are a few areas that require time-consuming manual intervention.

Recommended future enhancements such as leveraging standardized folder structures and naming conventions for SAS libraries, data sets, and outputs to programmatically identify and collect data sources. Another enhancement would be to develop a strategy to automate drawing the flow charts by using generic templates of common data source groupings. Programmatic automation of these steps in the process will reduce much of the planning and development time.

REFERENCES

- *Applying an Experimental GTL Feature to CONSORT Diagrams,* Shane Rosanbalm, PharmaSUG 2019 DV-021
- *Murach's HTML, XHTML, and CSS*, Anne Boehm, 2010, Mike Murach & Associates, Inc.

CONTACT INFORMATION

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

Jesse Pratt

jesse.pratt@ppdi.com

DISCLAIMER

The content of this paper are the works of the authors and do not necessarily represent the opinions, recommendations, or practices of PPD.