Automation of Appending Reports:  
A Sexy New Look  
Bill Coar, Axio Research, Seattle, WA

ABSTRACT
With a title like this, who won’t want to read the abstract? Now that I have your attention, the focus of this paper is the automation of creating a single, well-structured document for electronic review and distribution. Although numerous techniques exist to append reports, there is much room for improvement when creating a single document composed of multiple reports as well as a table of contents directly from SAS. A more attractive approach is considered here.

The proposed process takes advantage of external data used to track SAS programming, ODS Document, and Proc Document. Generation of the concatenated report requires two primary functions: identification of reports to be appended, and then the actual appending. Many companies already have infrastructure for tracking generation and testing individual reports produced using SAS. Provided SAS can communicate with the tracking component, a list of outputs with the desired sectioning can be obtained. This list is then used to read, restructure, and replay (PharmaSUG, 2013) using Proc Document.

This concatenation process is an extension of applications previously presented at PharmaSUG 2013 and WUSS 2013. It was developed for presenting data from clinical trials which are typically in the form of summary tables, listings, and figures (TLFs). It requires the use of ODS Document to create individual item stores during the initial creation of each TLF, and the use of Proc Document to manipulate and replay such item stores into a single, well-structured document with a hyperlinked table of contents and bookmarks.

An example showing how to prepare such a report will be presented using SAS 9.3 in a Windows environment.

INTRODUCTION
Although the setting for the application presented may be quite specific, it is not restricted to the pharmaceutical industry. Individual programs generate one or more reports (ie, TLFs) through the Output Delivery System (ODS) destination such as RTF or PDF. These programs are developed over time, and submitted in batch mode for production version of the reports. The final procedure for all summary tables and listings is Proc Report. Figures result from one of the SG procedures.

During the programming process, a tracking document is generally in place to easily find the stage of programming for each of the reports. While some outputs are being tested, others any only be in initial development, or not even started. This tracking document also allows a project manager to easily identify which reports are final. There is often other utility for such a file as well, such as this application provided the necessary columns exist.

The lengthy discussion below is as follows. After a motivating example based on [3] (a presentation from WUSS 2013), an example of an existing system to automate appending reports is shown as motivation for obtaining alternative approaches. Next, a review of the use of ODS Document and Proc Document is provided. Although many of the actual programming details are omitted, the key concepts of item stores as they relate to this application are presented. These are essential to understanding the incrementing steps of the proposed automated application. The actual automated process is described: reading in a program tracker to identify a list of reports, performing diagnostics, restructuring, and finally replaying. This is followed by a discussion of how this application can be used to automate the materials presented in the WUSS 2013. Eventually, this is followed by concluding remarks.

The techniques presented here are a review of the material presented in PharmaSUG 2014 [11] but with a more detailed discussion on item stores. The example subset listings presented in WUSS 2013 serve as a motivating example and desire for automation.

MOTIVATING EXAMPLE
In “Generation of Subset Listings” from WUSS 2013, the author presented a programming approach that satisfied two main requirements:

1. Generation of subset listings without any modification to existing (listing) code
2. Provide a highly structured single document for review and distribution.

Subset listings are simply (existing) listings run only on a subset of data. The subset can be anything from clinical site, a function of baseline characteristics, or simply a single patient. It was shown that this is possible under
reasonable constraints and the use of ODS Document and Proc Document. Below are two examples that were presented in the actual presentation based on [3]. The first is a simple example (Figure 1) of site listings (2 sites x 2 listings = 4 total listings in one document) in RTF format whereas the second is a more complex example of traditional patient profiles (30 patients x 16 listings = 480 listings in one document).

Figure 1: Site profile in RTF

Figure 2: Patient profile in PDF

The example code for Figure 1 is provided in [3], and is straightforward. However, the programming statements can rapidly become tedious and complex for a higher number of subsets and greater number of listings as is the case in
Figure 2. The application presented in this paper is a tool that greatly simplifies the programming required to create this highly structured single document with a tremendous amount of additional flexibility.

The following review of materials presented in [11] are intended for a more general setting of appending tables, listings, and figures. An extension of the general approach to apply to the subset listing application will be shown later in this paper. We proceed with the main discussion of appending tables, listings, and figures in the more general setting.

EXISTING METHODS

Historically, tables and listings were simply text files (LST) that included printer control language that allowed for nicely formatted printed output. With the introduction of ODS, tables and listings in RFT format have become more common. The SG procedures and Graph Template Language now allow for programmers to obtain highly custom graphs directly from SAS rather than other software such as R, SPlus®, or Excel®.

A number of methods to create a single document from a set of individual TLF files exist ([1],[6],[7],[8],[9],[10]) when individual files exist in LST, RTF, or PDF format. The more common setting is for reports to exist in RTF format. These techniques were reviewed in [1]. In the settings presented in [1], a typical process for automation might look like something in Figure 3.

Figure 3: Process flow for post-processing RTF files

Wrapped in some sweet SAS code, this existing method reads in a program tracker from Excel to identify a list of RTF files. The process relies of DDE to open Microsoft Word® and send Word Basic commands. This requires SAS license to communicate with Excel, and it also relies heavily on MS Word. A single document is obtained, though the table of contents is not actually an auto-generated Table of Contents from within Word. It is actually a simple table in Word with cross references to bookmarks throughout the document. The major drawback with this approach is the reliance on DDE. Although DDE can be a powerful tool, it has proven to be unstable in some server environments and the final file format can most certainly use improvement. Other methods of creating an appended report rely on Visual Basic, or modification of RTF tags and appending the text files.

Clearly there is a need for a new approach. The REPLAY approach presented in [2] is a technique for creating an appended report using current SAS software capabilities that independent of the user. Not only is the final output in this alternative approach more attractive, automation completely from within SAS makes it quite desirable.

REVIEW OF THE REPLAY APPROACH

This two-step process in Figure 4 was proposed in [2]. The first step of the process uses ODS Document to create item stores for each TLF. This is done in the initial creation of each TLF. These item stores house the instructions and data used by the procedures called within a block of ODS Document statements. The second step uses Proc Document to combine, restructure, and replay these item stores into a single document within an ODS destination.
Since all the item stores are replayed within a single ODS destination, the result is a single file that contains all the reports that encompass the TLFs.

A REVIEW OF FIGURE 4 COMPONENTS

An overview of item stores in the context of Figure 4 will allow the reader to better understand the automated process which is proposed in the next section, particularly the section on diagnostics and the section on restructuring into a single item store.

Recall that for each individual TLF, there is an existing block of ODS statements to create an RTF or PDF. If the additional ODS Document block is added, accompanied by (minor) program updates for style points, an item store will also be created when each program is executed. Item stores hold the data and instructions from the procedure used to create the report. They have procedure specific internal structures that may not make much intuitive sense individually, but structure becomes more apparent when put them together into a single restructured item store. The discussion presented here focuses on item stores created from Proc Report and SG procedures. For a more general discussion of item stores, see [4] and [5].

To see what is inside an item store created, the following code can be submitted:

```sas
proc document name=istore.l rand_closed;
  list / levels=all;
run;
quit;
```

**Sample Code 1: Item store from Proc Report**

The results are shown in **Output 1**, where we see the item store created when using Proc Report consists of directories and a table. To regenerate the report, a programmer needs to actually replay the item store.

```
Listing of: \istore.lrand_closed\
Order by: Insertion
Number of levels: All

<table>
<thead>
<tr>
<th>Obs Path</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1\Report#1</td>
<td>Dir</td>
</tr>
<tr>
<td>2\Report#1\Report#1</td>
<td>Dir</td>
</tr>
<tr>
<td>3\Report#1\Report#1\Report#1</td>
<td>Table</td>
</tr>
</tbody>
</table>
```

**Output 1: Item store from Proc Report**

Regarding graphics, the use of GTL adds often needed flexibility for producing custom graphics. Although the example below results from using GTL and Proc SGRender, they easily extend to other SG procedures as they all have similar structure. Suppose the code in **Sample Code 1** were resubmitted to list what is inside of an item store created from Proc SGRender, the output might look like **Output 2**.
Output 2: Item store from Proc SGRender with 4 Images

The item store in Output 2 was generated for a custom lab figure to assess patient values over time. Although the figure was created in a single RTF file, there were actually 4 images, one for each of 4 laboratory parameters of interest. Note here that this item store consists of directories and Graphs. For simpler figures that fit on a single page, the item store would only have a single directory and graph, both labeled Sgrender#1:SGRender#1.

The details of creating a single item store that contains all the TLFs are discussed in [2]. This involves reading and restructuring each individual item. This step essentially uses the COPY, MOVE, RENAME, and SETLABEL statements in Proc Document. These details of these statements are omitted here, but found in [2].

Individual item stores can be read into a single item store rather easily with a COPY command. For Output 3, item stores from 3 summary tables were read into a single item store.

Output 3: Three tables in a single item store

When looking at this combined item store, the directories of \Report#1, \Report#2, and \Report#3 can be thought of as parent directories. If we were to replay this (single) item store, the table of contents would appear to have 3 sections, one for each parent directory. If we want multiple tables or figures in the same section, they need to reside in the same parent directory. This is accomplished with restructuring.

Using the MOVE statement, a table from one parent directory can be moved to another. Output 4 shows the combined item store after the table in \Report#2 was moved to parent directory \Report#1.
Automation of Appending Reports: A Sexy New Look, continued

### Output 4: Moving a table to a different parent directory

Note that with the MOVE statement (as discussed in [2]) moves the directory \Report#2\Report#1 and the Table \Report#2\Report#1\Report#1 to the parent directory of \Report#1. This is noted in RED in Output 4. The remaining directory \Report#2 is not needed, and therefore removed with the DELETE statement. The result is now a restructured time store seen in **Output 5**.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Path</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\Report#1</td>
<td>Dir</td>
</tr>
<tr>
<td>2</td>
<td>\Report#1\Report#1</td>
<td>Dir</td>
</tr>
<tr>
<td>3</td>
<td>\Report#1\Report#1\Report#1</td>
<td>Table</td>
</tr>
<tr>
<td>4</td>
<td>\Report#1\Report#2</td>
<td>Dir</td>
</tr>
<tr>
<td>5</td>
<td>\Report#1\Report#2\Report#1</td>
<td>Table</td>
</tr>
<tr>
<td>6</td>
<td>\Report#2</td>
<td>Dir</td>
</tr>
<tr>
<td>7</td>
<td>\Report#3</td>
<td>Dir</td>
</tr>
<tr>
<td>8</td>
<td>\Report#3\Report#1</td>
<td>Dir</td>
</tr>
<tr>
<td>9</td>
<td>\Report#3\Report#1\Report#1</td>
<td>Table</td>
</tr>
</tbody>
</table>

### Output 5: Restructured item store

When the restructured item store in **Output 5** is replayed, the table of contents now has only 2 sections. The first section would have two tables whereas the second section has only one table. In the item store, the sections are identified by \Report#1 and \Report#3. Note that these are just labels for the directories. Although they are labeled as Report#1 and Report#3, they just as easily can be renamed to something else. The use of the RENAME statement becomes important when a programmer wants to include figures in a parent folder that also includes tables from Proc Report.

Based on the above examples, it is clear that the proposed process deals with *directories, reports, and graphs*. Although not immediately obvious in the above Outputs 2-5, it is seen that SAS (predictably) increments as additional directories and reports/tables/figures are added. *This incrementing is essential when automating the creation of a single restructured report.*

A more complicated restructured item store can be seen in **Output 6**, which is used to further illustrate the necessity to automate the process of *read and restructure*. The final output based on the Output 6 item store is found in...
Examples 1 and 2. This item store (and output) has multiple sections, and some sections include combinations of tables, listings, and figures.

```
Listing of 'Work.App'
Order by: Insertion
Number of levels: All
```

<table>
<thead>
<tr>
<th>ObsPath</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report#1</td>
<td>Dir</td>
</tr>
<tr>
<td>Report#1</td>
<td>Dir</td>
</tr>
<tr>
<td>Report#1</td>
<td>Table</td>
</tr>
<tr>
<td>Report#2</td>
<td>Dir</td>
</tr>
<tr>
<td>Report#2</td>
<td>Dir</td>
</tr>
<tr>
<td>Report#2</td>
<td>Table</td>
</tr>
<tr>
<td>Report#8</td>
<td>Dir</td>
</tr>
<tr>
<td>Report#8</td>
<td>Dir</td>
</tr>
<tr>
<td>Report#8\Report#1\Report#1</td>
<td>Table</td>
</tr>
<tr>
<td>Report#8\SGRender#1</td>
<td>Graph</td>
</tr>
<tr>
<td>Report#8\SGRender#2</td>
<td>Graph</td>
</tr>
<tr>
<td>Report#8\SGRender#3</td>
<td>Graph</td>
</tr>
<tr>
<td>Report#8\SGRender#4</td>
<td>Graph</td>
</tr>
</tbody>
</table>

```
 | Report#13\Report#1\Report#1 | Table |
| Report#13\SGRender#1 | Graph |
| Report#13\SGRender#2 | Graph |
| Report#13\SGRender#3 | Graph |
| Report#13\SGRender#4 | Graph |
```

Output 6: More complex restructured item store

Take note of the directory numbers circled in RED. First, the left-most directory level is the same within each section. Second, the number of this directory is a function of the number of “Reports” (either tables or graphs) previously read into the structured item store. For example, the second section is labeled “Report#2” simply because the first section only had 1 report (ie, one table) in it. The third section is labeled Report#7. Even though this is the third section, the report label is Report#7 because the first report in the third section is actually the 7th report (table or graph) read into the restructured item store.

As seen in the fourth (and fifth) sections, a section can have tables, listings, and/or figures. In the above example, the fourth section (Report#8) has one listing and four figures. Recall both tables and listings come from Proc Report, so the contents of the item store are labeled “Table”, regardless if it is a summary table or listing. Output 6 alone does not suggest Report#8 contains a listing and 4 graphs, but this becomes clear when this item store is replayed.

Even though only the first element in section 4 actually comes from Proc Report, incrementing continues to occur with respect to the label of the next section (Report#13). This is by design of the application. Recognizing that the left-most directory level is really just a label that gets incremented, the application renames the directories of figures to “Report”.

SAMPLE OUTPUT

To clear the mind from the above discussion on incrementing, it is useful to view the output from replaying the first 4 sections of the item store in Output 6 into a (single) RTF file and a (single) PDF.
Example 1: PDF from replaying four sections in the Output 6 item store

When using ODS to output directly to PDF, a highly structured (and searchable) PDF file is created with all of the individual TLFs in the desired sectioning. Although there is no Table of Contents, the bookmarks on the left-hand side provide the similar information and since the bookmarks are hyperlinks to each report, navigation throughout the document is quite easy. The absence of summary statistics in the demographics table in Example 1 is by design for this paper since there data are proprietary.

The corresponding output in RTF format is seen in Example 2.

Example 2: RTF from replaying four sections in the Output 3 item store

When using RTF, a programmer can obtain a highly customizable (hyperlinked) table of contents. It is recommended that these RTF files actually be re-saved as Word documents. Although both Example 1 and Example 2 have desirable formats, creation of the restructured single item store in Output 6 may seem intimidating, especially since the sectioning is likely to change from among projects, or even reports within a project. Automation of these steps provides tremendous efficiencies.
ODSPD APPROACH

The proposed macro based approach, ODS/Proc Document (ODSPD), automates the replay approach presented in [2]. As seen in Figure 3, the process of generating the single document can be simplified.

![Figure 5: A simplified approach using ODS Document and Proc Document](image)

The proposed approach still requires the ability to communicate with a program tracker, but it eliminates the need for DDE. It allows a programmer to create both RTF and PDF without the need for additional software. Note that SAS will create an RTF file with the necessary components for a hyperlinked Table of Contents, but the user must actually open the file and update the field to populate it. A PDF with bookmarks can be created directly from SAS without the need to purchase a license to PDF writer software.

The program flow with the ODSPD approach is straightforward: read in a list of TLFs from a program tracker, perform some diagnostics (checkis.sas), build the restructured (single) item store (makeapprtp.sas) with desired sectioning, and then replay the desired sections (rplapprpt.sas). Although the actual code is not provided in this paper, it can be made available by request.

READING IN THE PROGRAM TRACKER

Most programming departments seem to have external files used to track program development and testing. Such a “tracker” can be used for our purposes as well with a few additional columns. The basic requirements are:

- One record per report element defined as a single table, listing, or graph
- (Numeric) Column for sorting sections
- (Character) Column for section names
- Item store name
- (Numeric) Column for sorting within section
- SAS procedure used
- Internal item store sequence number (should it contain multiple tables or graphs)

An example of the tracker used in the previous example is shown below.

<table>
<thead>
<tr>
<th>Section Number</th>
<th>Section Name</th>
<th>TLF Type</th>
<th>TLF Number</th>
<th>ItemStore Name</th>
<th>Item ID</th>
<th>Proc Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Randomization</td>
<td>Listing</td>
<td>1</td>
<td>lrand_closed</td>
<td>1</td>
<td>Report</td>
</tr>
<tr>
<td>1</td>
<td>Tables</td>
<td>Table</td>
<td>1</td>
<td>rdemog_closed</td>
<td>1</td>
<td>Report</td>
</tr>
<tr>
<td>1</td>
<td>Tables</td>
<td>Table</td>
<td>2</td>
<td>raepl_closed</td>
<td>1</td>
<td>Report</td>
</tr>
<tr>
<td>1</td>
<td>Tables</td>
<td>Table</td>
<td>3</td>
<td>rsaep1_closed</td>
<td>1</td>
<td>Report</td>
</tr>
<tr>
<td>1</td>
<td>Tables</td>
<td>Table</td>
<td>4</td>
<td>rhema_closed</td>
<td>1</td>
<td>Report</td>
</tr>
<tr>
<td>1</td>
<td>Tables</td>
<td>Table</td>
<td>5</td>
<td>rchem_closed</td>
<td>1</td>
<td>Report</td>
</tr>
<tr>
<td>2</td>
<td>Listings</td>
<td>Listing</td>
<td>1</td>
<td>lsae_closed</td>
<td>1</td>
<td>Report</td>
</tr>
<tr>
<td>3</td>
<td>Hematology Data</td>
<td>Listing</td>
<td>1</td>
<td>lhemalab_closed</td>
<td>1</td>
<td>Report</td>
</tr>
<tr>
<td>3</td>
<td>Hematology Data</td>
<td>Figure</td>
<td>2</td>
<td>fhemabpbs_closed</td>
<td>1</td>
<td>Srender</td>
</tr>
<tr>
<td>3</td>
<td>Hematology Data</td>
<td>Figure</td>
<td>3</td>
<td>fhemabpbs_closed</td>
<td>2</td>
<td>Srender</td>
</tr>
<tr>
<td>3</td>
<td>Hematology Data</td>
<td>Figure</td>
<td>4</td>
<td>fhemabpbs_closed</td>
<td>3</td>
<td>Srender</td>
</tr>
</tbody>
</table>

**Example 3: Tracker in Excel**

In order to know verify what is inside the item store for diagnostics and restructuring, the application needs to know...
what procedure created the item store, and what report (or graph) number it will be. This has been more useful for
graphs, and in particular, graphs of laboratory data done by parameter. The graphing is not actually performed using
BY VARIABLE processing. Such processes produces items stores with a different structure. For this application,
one-at-a-time graphics were utilized which results in a straightforward item store structure containing multiple graphs.

As seen in the red box in Example 3, we can expect to find the first listing in section 3 to be associated with
Report#1, the first figure to be associated with Sgrender#1, the second figure is associated with Sgrender#2, and so
forth because it is known that the item store from creation of the hematology figure had multiple graphs inside. With
an understanding of the individual item store structures, a programmer can then predict what should be in the actual
item store in the diagnostic step of the application.

DIAGNOSTIC: CHECKIS.SAS

When Proc Document experiences problems, uninformative errors occur leaving the programmer frustrated and
almost certainly willing to give up. Throughout the development of ODS Document and Proc Document techniques, it
has become clear that most of these problems are due to missing or incomplete item stores. Such is the case when
programmers overlook warnings or errors in the log files when an individual item store was created. Thus, a
diagnostic step has proven to be quite useful.

The macro to perform diagnostics has the following parameters used to predict if an item store specified in the tracker
actually exists:

- **INDATA**: Input dataset (presumably from the tracker)
- **INAME**: Library name that contains the item stores
- **ISNAME**: Character variable containing the name of each item store
- **PROCTYPE**: Character variable containing the name of the procedure that created the item store
- **IDNUM**: ID number internal to each item store. If multiple reports or graphs are within a single item
  store, use this IDNUM to identify which internal element is to be used

The general flow of the diagnostic is to first obtain a list of item stores that exist in the specified library as shown in
Example 4(a). This can be used to verify that the requested item store files in the tracker actually exist. The macro
then looks to see if the requested elements (ie, Report#1, Sgrender#1, etc.) exist inside using item store properties.

```
ods table documents=zz_isdocs;
proc document;
   doc lib=&INAME;
run;
quilt;
```

(a)

```
ods table properties=zz_isprop;
proc document name=&THISIS;
list / levels=all details;
run;
quilt;
```

(b)

Example 4: Code snippets from checkis.sas

The key concept is the use of Documents and Properties ODS tables to obtain the information to perform the
diagnostic. This information allows a programmer to see if the item stores and elements needed for the report actually
exist. Printed output can be used for immediate reference should errors occur in the next step, the restructuring
process.

RESTRUCTURE: MAKEAPPRPT.SAS

Of all of the discussion in the document, this macro is actually the backbone of the application. Since its
implementation, creation of restructured item stores has become almost seamless. The macro has the following
parameters in addition to those used in CHECKIS.SAS:

- **SECNUM**: (Numeric) variable to identify and sort different sections
- **SECNAME**: Character variable that has the name of each section. Section name assumed to be
  the same for all records within a given section number.
- **SRTNUM**: Numeric variable for sorting within each section (such as the table number)

It does not take long for a programmer to appreciate the value of automating the creation of the restructured item
store. The specific details of restructuring are discussed in [2]. The primary key for automation is recognizing SAS’s
automatic incrementing of directory and element numbers as they are read and restructured into a single item store.
This was why incrementing was emphasized earlier. Incrementing macro variables within loops handles this quite
The flow of the restructuring is to first see if IDNUM is specified. This is not required if all item stores are restricted to a single report or graph, as is the case with some programming styles. Should this be the case, the IDNUM macro parameter is not necessary. Thus far, the utility of IDNUM has come from figures that require multiple pages of images.

Creation of macro variables and looping are key concepts of the restructuring macro. Macro variables are generated for lists and numbers of elements associated with each list. Lists are determined for different section names, lists of item stores within sections, and a list of elements (Report#, SGrender#, etc.) within each section. Lists in the form of macro variables are a natural fit for looping through each section with the necessary incrementing.

Example code snippets from MAKEAPPRPT.SAS macro are shown below.

Example 5: Snippet of general looping in makeapprpt.sas

As seen in Example 5, looping occurs over the number of sections and the number of elements (reports or graphs) within each section. Information from each list is identified for a given element within a section.

Example 6: Snippet of restructuring in makeapprpt.sas

Recall each section can be identified by the left-most directly label. This directory label, denoted by the macro variable THISII, is the same for all TLFs within a section. This only gets incremented with the introduction of a new section. There is also incrementing with respect to each element read into the restructured item store, denoted by THISI. The incrementing (actually re-defining) of these two macro variables is consistent with that described in Output 6.

The above snippet of code in Example 6 includes a conditional statement required for figures. Since tables and listings tend to be more prevalent that graphs, and both are labeled (by default) by "Report", any element that is read in that is not labeled as "Report" is renamed accordingly. It is this step that allows tables, listings, and graphs within the same section.

Lastly, a few additional programming statements are needed for the first element within a new section. The first is to increment THISII, and is trivial. The second is with respect to the RENAME and MOVE statements. If the first
element in a new section is from Proc Report, one only needs to COPY the (single) item store in the restructured item store. However, if the first element is a figure, a RENAME statement is required. This is due to the difference between item store structures from Proc Report and the SG Procedures. The above sample code is a snippet from makeapprpt.sas that does not take into consideration if the element is the first in a (new) section.

The last essential component of makeapprpt.sas is the creation of a global macro variable containing a list to identify all of the sections within the restructured item store. What may not be obvious to this point is that the section label numbers (such as Report#1, Report#2, Report#7, and Report#13 in Output 6) will change as item stores are added or removed to each section, or if entire sections are added or removed. To accommodate this, an additional global macro variable containing a list of sections is defined in makeapprpt.sas.

With the restructured item store easily obtained, the programmer simply needs to replay all or some of the desired sections.

REPLAY: RPLAPPRT.SAS

The last step in the automated process is to replay either the entire report, or individual sections of the report. When this is done inside of the infamous ODS sandwich, the result is a single well-structured file with all of the individual TLFs in the user-specified sections. Compared to the other macros in the application this is trivial, with only two macro parameters:

- INAME: Name of the item store to be replayed
- RLIST: Character string of sections of an item store to be replayed

The general flow is a simple loop through the various sections of the report, replaying each section as seen in Example 7. Although the development of this macro was initially for replaying in this application, it is in fact a stand-alone macro, and is now used in other applications.

Example 7: Snippet from rplapprpt.sas to replay sections of a report

As long as the macro call is within an ODS block to the desired (RTF or PDF) destination, a single, well-structured document is obtained.

REVISIT THE SUBSET LISTING APPLICATION

As mentioned in the motivating example, the read and restructure step can rapidly become complicated for cases with a large number of subsets and/or large number of listings. Even after initial programming is in place, making a minor change such as removing or adding one listing to each subset section can be quite tedious. The ODSPD system presented here can easily be utilized for the subset listing application.

The first thing to recognize is that the macro based system ultimately relies on a SAS dataset. If a SAS dataset can be created with the appropriate structure, the ODSPD can simply run as usual. Consider the output in Figure 2: Patient profiles in PDF. Although not all are shown, this document contains 16 listings x 30 patients=480 item stores that are combined to create a single document. If we assume 3-5 programming statements are required to read and restructure each item store, a programmer easily exceeds 1000 lines of code (or 1 KLOC for the programming gurus among us).

Recall that the subset listing takes existing programs and re-runs them on subsets of data without any modification. In this process, each output and item store are strategically named that maps a file id to an individual subset. A simple approach is to sort/select unique subsets and create a sequence number associated with each. For example:
Table 1: Mapping of patient ID to File ID

<table>
<thead>
<tr>
<th>Patient Number</th>
<th>Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>003-204</td>
<td>001</td>
</tr>
<tr>
<td>009-201</td>
<td>002</td>
</tr>
<tr>
<td>011-202</td>
<td>003</td>
</tr>
<tr>
<td>Etc</td>
<td></td>
</tr>
</tbody>
</table>

The resulting output files after each program is run on each subset will have the naming convention
original_file_name_fileid. Knowing this, a dataset can be prepared to call the ODSPD macros.

**STEP 1 – BASE OUTPUT NAMES FROM TRACKER**

Since the listing programs are assumed to already exist in the subset listing application, it is also assumed that they will have entries in the existing program tracker. The first step is to read in the records to identify the base listings into SAS. For the output in Figure 2, sixteen (16) records are read in from the program tracker.

**STEP 2 – IDENTIFY UNIQUE SUBSETS**

Create a dataset with one record per unique subset. In this example, a dataset with 30 records is obtained. This is sorted by patient number and a sequence number corresponding to the FILEID is derived. This sequence number must be consistent with that used when executing the listing programs on each subset.

**STEP 3 – CARTESIAN PRODUCT**

A Cartesian product is used to join these datasets from Steps 1 and 2. The resulting dataset now has 480 records.

**STEP 4 – PREPARATION FOR MACRO CALLS**

The final preparation requires a data step to create several fields needed for the macro calls: section name, section number for sorting, and item store name.

- The section names from the tracker are ignored. They are redefined in this data step. The section label in the TOC is the value of each unique subset. The section sorting variable is also redefined so that it is based on the unique subsets.
- The file name for each item store is derived based on the original item store name with the FILEID appended to it (original_item_store_name_fileid). For example, l_rand_001 is the randomization listing for the first patient, l_demog_001 is the demographics file for the first patient, l_rand_002 is the randomization listing for the second patient, l_demog_002 is the demographics for the second patient, etc.

All other components from the tracker remain the same. At this point, the programmer can simply call the ODSPD macros as usual. With that, the actual SAS program that generates the single file for review and distribution is greatly simplified. Adding or removing individual listings, or changing the sort of the listings, is easily done with either modifications to the tracker or programming statements in STEP 4.

**CONCLUDING REMARKS**

The reader is congratulated for making it this far into the discussion. There is a lot to discuss to even begin to understand the totality of the process. The application has enabled automation that allows for great flexibility with sectioning, and even allowing tables, listings, and figures within the same section. The user can easily add or remove report elements, or create additional sections. The application is independent of the user, allows for both RTF, PDF, or both, and does not require knowledge much beyond SAS. It has natural extensions as seen with its application to the subset listings.

As with many applications, this continues to be a work in progress. Although automation has only recently been achieved, the techniques using ODS Document and Proc Document have been successfully utilized at Axio for over two years. The techniques allow us to provide a single well-structured, hyperlinked, searchable document for distribution and electronic review with minimal to no reliance on other software.
REFERENCES

[5] Lawhorn , B. 2011, Let’s Give ‘Em Something to TOC about: Transforming the Table of Contents of Your PDF File, SAS® Global Forum

ACKNOWLEDGMENTS

The author would like to thank the programmers and statisticians at Axio for their discussions in developing the techniques contributing to this paper. The author would also like to acknowledge the many programmers publishing information on the internet, and apologizes to those not properly referenced above, citing the inherent nature of internet searching which makes it difficult to track so many individual ideas.

CONTACT INFORMATION

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

William Coar, PhD
Biostatistician/Director of Central Region Operations
Axio Research, LLC
Seattle, WA 98121
Email: williamc@axioresearch.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

Other brand and product names are trademarks of their respective companies.