Meta Data + Files = Automated Status Reports

Frank DiIorio, CodeCrafters, Inc., Chapel Hill NC
George DeMuth, Stat-Tech Services, LLC, Chapel Hill NC
Michael DeSpirito, Chapel Hill, NC

Abstract

Let's be realistic – no one looks forward to writing status reports. Even if everything is up to date and fine, reporting this good news still requires time and effort. Couple this with the typical programmer's desire to automate everything in sight, and you have … %STATUS, a tool to generate project status reports.

This paper describes the need for and evolution of the program, and discusses the file structures, and coding techniques we employed. We focus on several key ingredients that make programs like %STATUS feasible, successful, and relatively easy to code. The first is meta data. Our program, and many others in the project, uses this high-level "data about data" to determine what tables and listings should be generated. Once this is known, we know what output / reports should be generated. The second key ingredient of the program, ODS-generated HTML, can then be used to communicate what has actually been produced.

The reader should come away from this presentation with an appreciation of the utility of meta data, an understanding of how to convert this data into a useful and visually appealing format, and a knowledge of some basic ODS SAS programming techniques. The paper will probably not, however, increase the reader's appetite to write status reports.

Introduction and Organization

Background

A critical aspect of any project, regardless of size, is to keep participants informed of progress toward goals. In the scenario that created the need for this program, we had to generate a set of tables, listings, and graphs for a client. During the course of the project, we had to keep constant track of what output had been created and what issues may be affecting production.

A traditional solution is to create a file (spreadsheet or similar document) with a complete list of what has to be produced. Then, as milestones are reached and validated output created, mark the items' completion status.

Such labor-intensive, manual processes do not encourage frequent updating, and they are error-prone. This, combined with the uneasy realization that yes, we're all a bunch of tech-heads looking for a new challenge, resulted in the program described in this paper. It relies on the predictability of a project's directory structure and project meta data to create a web page with hyperlinks to programs and output, and allows entry of comments to enhance communication between team members.

Organization

The paper is organized as follows:

- The first sections describe the high-level pieces of the system: predictable, logical file organization and meta data.
- We then use the strengths of these system features and describe how they are processed. During this discussion, we elaborate on not only the approach that we took, but alternatives that were tried but rejected. We show and discuss two sample reports.
- Finally, we present a series of technical "gotcha's," or parts of the program that presented technical challenges. These focus mostly on generating HTML and the differences in the REPORT procedure's creation of "plain ASCII" and HTML files.

Caveats

The code that is actually in the %STATUS macro is more complex and functional than what is presented in this paper. In general, we try to give a high-level overview of what is involved in writing the application, then present some coding-specific examples for issues that we think are most relevant to newcomers to HTML.

Due to space limitations, we do *not* represent this as an "application primer" containing a complete description of the program's logic and coding. We encourage readers who are interested in these matters to contact us directly (see "Contact Information," at the end of the paper).

Ingredient One: Predictability

A utility such as the one described in this paper becomes fairly easy to write if there is an element of predictability in its data sources. Our work consists largely of the production of tables and listings in support of clinical trials for drugs and devices. This lends itself to a fairly standardized directory structure – data is provided by the client and is validated. We create reports in the form of listings, tabular summaries, and graphs. The directory structure associated with a client's study reflects the life cycle of the data. A simplified structure is presented in **Exhibit 1** (Exhibits that are not in-line are presented following the body of this paper).

Notice a few relevant features of the display:

- Different aspects of the data and report production life cycle are addressed by the directory structure. There are Production and Test areas, different directories for different types of data. More to the point for this paper, there are subdirectories for different types of output – Graphs, Listings, and Tables. If we want to manually or programmatically go to a particular type

of output, the organization and naming convention makes it easy to do so.

- The Tables directory is highlighted, and its contents are displayed in the right-hand pane of the window. Files have extensions of TBL (the extension used by the client for plain text output) and RTF (the standard extension for Microsoft Rich Text Format output). Each table – demog, enroll, etc. – is provided to the client in these two file formats. Thus a complete set of tables will have RTF and TBL files for each of "n" tables.

But how do we know what "n" is? This is where meta data comes into play.

Ingredient Two: Meta Data

Meta data, classically defined as "data about data," has for some time been part of the SAS System and has mightily influenced the scope and design of SAS-based systems. The "dictionary tables" available in the SQL procedure and their associated views in the SASHELP directory contain a wealth of data about SAS data sets, options, macro variables, and the like. Their organization is predictable and continually updated during execution of a SAS program. They provide information to the programmer that would otherwise be very difficult, if not impossible, to acquire.

In the same spirit of predictability and data abstraction, we can create meta data to describe processes to programs. The data can be in virtually any format and granularity. In our case, the meta data is a plain ASCII file whose granularity is information about parts of a table. **Exhibit 2**, below, has part of the meta data for our sample client.

Exhibit 2: Sample Meta Data

```
progname=enroll
ref=tbl
t5=Table 1
t6=Subject Enrollment Profile

progname=demog
ref=tbl
t5=Table 2
t6=Demographics, Screening Data
f1=[1] P-values are Cochran Mantel Haenszel

progname=asthma_hist
ref=tbl
t5=Table 3
t6=Medical History
c1=!Site 002 queries still outstanding
c2=Site 034 queries completed as of May 21st

progname=asthma_hist
ref=tbl
t5=Table 4
t6=Asthma History at Screening
t7=Enrolled Subjects

progname=asthma_trt_hist
ref=tbl
t5=Table 5
t6=Asthma Treatment History at Screening
t7=Enrolled Subjects
c1=!No treatment history data is available yet
```

The format is as effective as it is simple. Each line is a name-value pairing, separated by an equal sign (=). The set of valid names is known to the programs that use the meta data. Likewise, the order of names is predictable. PROGNAME begins the set of name-value pairs for a reporting entity, and it is followed by REF. The Tx and Fx names are used for report titles and footnotes, and the Cx names are reserved for comments. The latter are items containing information that needs to be shared, but does not necessarily need to appear on output generated for the client.

There is considerable value added to this method. In addition to being easy to maintain, it encourages consistent and accurate reporting. Changes in wording, program naming, and the like can be made quickly and reliably. Rather than change "Con Med" to "Concomitant Medications" in five separate files, for example, we'd simply make the change in one location – the meta data. All that's required is that reporting programs be "meta data aware" and actually use the macros that generate the titles and footnotes and otherwise process the data.

Let's look at the last group in Exhibit 2 and see what it tells us, bearing in mind that there are several naming and procedural standards at work:

- The PROGNAME value means that there will be a program named asthma_trt_hist_tbl.sas in the Source\SAS directory. The REF value of TBL was used in properly identifying the program (a separate directory under Source\SAS for each output type was considered, but the client requested this structure). TBL also tells us that we are in the description of a *table's* output, and so the TBL and RTF files will reside in the Output\Tables directory. In fact, we can now construct the names of the files we expect to find:

  `…\Output\Tables\asthma_trt_hist.tbl`

  `…\Output\Tables\asthma_trt_hist.rtf`

- The T5 through T7 values are the values that the table will use for some of its report lines (title lines 1 through 4 are standardized throughout the study and are handled elsewhere by a reporting macro). There can be up to 10 T parameters. Keywords beginning with "F" are, not surprisingly, footnotes.
- C1 is a comment used by our status report application. The value starts with an exclamation point. This tells the program to display the comment larger than normal, and in red. Otherwise, it will be shown in the same point size as other text, and colored black. There can be up to 10 C parameters.

The implementation of the Cx names shows the power of the meta data concept. The need for these comments was not identified when the meta data was designed. Since the programs that used the data only looked for specific names (PROGNAME, Tx, etc.), the inclusion of a new set of names went unnoticed by the existing program base. Because these programs did not "complain" about unexpected keywords, we could introduce the Cx names for our use without disturbing the other programs. Had that not been

possible, of course, we could maintain separate meta data containing only comments, but it seemed best to avoid a proliferation of such files.

Nice Ingredients. How Will We Use Them?

Let's consider what we have at this point in the discussion that is relevant for the application being discussed in this paper:

- "Actual" files. Because the directory structure is clearly organized, we can readily locate tables, listings, or graphs. Each directory will have a specific set of file types – TBL and RTF for the Tables directory, GIF for the Graphs directory, and so on.
- "Expected" files. The meta data contains the complete enumeration of report names and types for a project. Thus we can use it to produce a listing of what constitutes a complete set of files. In the example from the previous section, we know that if REF is "tbl" that we have an output type of table. And we know from our knowledge of the project that we should produce two files, namely:

  ```
  asthma_trt_hist.tbl
  asthma_trt_hist.rtf
  ```

  Likewise, since the SAS program name matches the PROGNAME value and since the location of SAS programs is predictable, we can construct the names of SAS programs and output that we'd expect to produce:

  ```
  ...\Source\SAS\asthma_trt_hist_tbl.sas
  ...\Source\SAS\asthma_trt_hist_tbl.log
  ...\Source\SAS\asthma_trt_hist_tbl.lst
  ```

Recall that the key to our status reporting is to say, in effect, "here's what we have to do, here's what we've done, and here are some related comments." We now have all the pieces required to produce a report.

"What we have to do" is in the meta data, "what we've done" is assessed by looking in the appropriate directories, and "related comments" are the Cx fields in the meta data. All that remains is to put the pieces together in a readable, attractive package.

Blending the Ingredients

The program that creates the report, while not short, at least has the advantage of being able to exploit the directory and file-naming practices that we've described above. In this section, we describe the program's design, focusing on *what* the program does, rather than *how* it does it. Details of some of the trickier and less obvious code are discussed later.

Intialization

The entire program is contained in a macro. Among the keyword parameters are:

- BASE – the root directory of the project we're reporting
- TYPE – the type of output to report on – TBL, LST, or GRAPH
- TITLE1, TITLE2 – Title lines used by PROC REPORT as it writes the HTML.

Identify "Actual" Report Files (Data Set ACTUAL)

Once we know where to look (derived from the BASE and TYPE parameters), we can read the directory to see what's present. The program is run in a Windows platform, so a piped DIR command is used to capture the directory. We create one observation per file name. Among the variables are the file name and one or more variables matching the requirements of our report type. Thus if the TYPE parameter was TBL, the processed DIR output data set would contain variables TBL and RTF, since these are the two file types created by plain text ("TBL") reports. The variables contain the HTML (HREF command) needed for a hyperlink to the file.

We could have taken another approach. Rather than run the DIR command and massage its output, we could have tested for the presence of the file(s) we expected. After all, we know what directory to look in, and we could read the meta data to build all expected names. Combining these pieces into calls to the EXIST function would not be difficult, and would require less coding than the approach we implemented.

The problematic aspect of this approach is that it would only show what was produced from what was *expected*. That is, the calls to EXIST would be driven by the meta data. They could not, by definition, identify files in the directory that were *not* expected. These could include files with old names, reports that became obsolete, and so on. Since these directories were to be shipped to the client, some automated means of identifying stragglers would be helpful. Such identification is not possible if we relied only on the meta data.

Read the Meta Data (Data Set META)

Next, we read the meta data. This is a simple process. We read the text file with the data, relying on its organization to make decisions about whether and when to output an observation (recall PROGNAME begins a report type, and it is followed by REF). We accumulate Tx and Cx values within a program-ref combination, creating one observation per PROGNAME-REF, with variables T1-T10 and C1-C10.

Look for Program Artifacts (Data Set ARTIFACTS)

The next, and final, group of files is program-related. We create a data set whose observations correspond to the program names we expect. For each program, we test for the presence of SAS program, Log, and listing files in the source directory. Among the variables in this data set are: the SAS, LOG, and LST. Like the "actually created" RTF, TBL, and other variables described above, these variables contain code for HTML hyperlinks.

Combine the Pieces

Now we can combine data sets ACTUAL, META, and ARTIFACTS. Using IN data set parameters, we can identify various situations:

in_actual & in_meta	Files that we expected
in_actual & ^in_meta	Unexpected files
in_meta & ^in_artifact	Expected, but no program artifacts (program, log, or list)

This DATA step also builds the variables that are actually processed by PROC REPORT. Recall that there are a varying number of Tx and Cx lines in the meta data. Recall, also, that there may or may not be values for report variables such as TBL and RTF, or for artifact variables such as SAS, LOG, and LST. In this DATA step, we build new variables that are the concatenation some of this information:

- TITLE_LINES concatenates all non-blank title variables (Tx). It also identifies the source for each line (the "x" in Tx). Then, if any comments are present, inserts the text "Comments follow", followed by any non-blank comments. If a comment had to be displayed with emphasis (began with a "!"), we insert the required HTML to display the text larger and in red. To make the report readable and thus useful, we had to take pains to format it attractively. This presented some technical issues, which will be addressed in a later section.
- SAS_LINES concatenates the HREFs for non-blank values of SAS, LOG, and LST. As with TITLE_LINES, making the presentation of this information visually appealing presented some challenges. Coding examples will be presented in a later section.

(Finally!) Create the Report

By this point, all the hard work has been done. The data sets have been combined, anomalies have been identified, and HTML has been built. All that remains is to identify the report style (HTML) and location to the Output Display System (ODS), and set other ODS options. The PROC REPORT code is "pure vanilla" – no tricks or advanced techniques are required. All that we need is simple application of the COLUMN and DEFINE statements.

We run REPORT twice, once for items that are expected, given the meta data, and once for unexpected output. The output from the first pass of our sample data is shown in **Exhibit 3**, at the end of this paper. The notes below are keyed to the numbers in the Exhibit's call-outs:

[1] We provide a link to the title file by coding a header spanning the leftmost columns. The line separator "~" was specified by the SPLIT parameter in the PROC REPORT statement:

```
column ("Title file information~<a
href='&base.\utility\titles.txt'>(click here
to view title file)</a>" statement continues
…
```

[2] We date-time stamp the report with another header spanning multiple columns.

[3] Titles are displayed in ascending order (T5, T6, etc.), each title starting on a new line. The title number is displayed between the "#" characters, e.g., #5#. Skipped values are not displayed. That is, if the meta data contained values for T5 and T7, we would not print a line for T6.

[4] Comments follow titles. We print one comment per line. In this case, the person editing the meta data wanted to draw attention to outstanding issues with Site 002. Since the meta data value began with "!" (see Exhibit 2, above), the program adds HTML to increase the point size and change the color to red. Once the line is displayed, we revert to the standard point size and text color.

[5] The Source, Table, and RTF columns identify files that we found in the programmatically-determined directories. Thus we see that program ENROLL has a SAS program and Log, but no listing file. We also see that both the TBL (plain ASCII) and RTF files were located. Clicking on any of these items results in displaying the file with the associated viewer (the application associated with the extensions SAS, LOG, LST, TBL, and RTF).

We see at a glance that items 3 and 5 are incomplete. Item 3, asthma_hist, does not yet have a table or RTF file. Item 5, asthma_trt_hist, appears not to have started – no SAS program or output, nor any table or RTF.

Exhibit 3 addresses the cases where we have some, part, or none of the output that we expected, given our reading of the meta data and the directories. Recall that we deliberately read all of the report directory's files in order to identify anomalies (items that we found, but did not expect). **Exhibit 4**, at the end of this paper, shows how the program reacts to finding TEST_PGM.TBL in the tables directory. Note that the left-most columns and headers are different than those in Exhibit 3.

Technical Gotcha's

Most of the program is pretty straightforward Base SAS coding – DATA steps that read reliably-formatted ASCII files, some macro language coding, and PROC REPORT with some attached styles. Had this been a simple hardcopy, LST file-only application, it would probably not even be worthy of being a paper at this conference.

HTML provides the value-added – the hyperlinks make it a simple but effective interface to key project files; the ability to change font size and color lets us show areas of concern at a glance; and the ability to control color, font, and other esthetic aspects of the listing make it visually appealing. This may seem like a "sizzle versus steak" discussion, but it does seem that if information is presented in an attractive, easy to read format, it has a better chance of being utilized than the same content indifferently packaged.

Here are some of the low-level coding issues that we identified during the program's development. The order of their presentation should suggest nothing about their relative importance or the amount of time we spent before we

overcame them.

Creating HREFs

Hyperlinks in the report provide quick and reliable access to project files (SAS programs and output, reports, and graphs). Each item for a program (the SAS program, Log, listing, and report files) was assigned to a variable. This is a somewhat syntactically challenging process. To illustrate, suppose we read the macro's TYPE parameter and assigned the value:

```
C:\SESUG Sample Clent\Apps\SRS\Prod\Output\Tables
```

to macro variable DIR at the beginning of processing. We then execute this command:

```
filename allfile pipe "dir/b &dir\*.*";
```

A DATA step reads this line from ALLFILE and assigns it to variable LINE:

```
enroll.tbl
```

Variable TBL is created as follows (line breaks are inserted to improve legibility):

```
tbl = "<a href='&dir.\"  ||
      trim(line) ||
      "'>" ||
      trim(scan(line,1)) ||
      "</a>"
```

The value of TBL becomes:

```
<a href='C:\SESUG Sample Clent\Apps\SRS\Prod\
Output\Tables\enroll.tbl>enroll</a>
```

When it is displayed by PROC REPORT as an HTML file, it is simply

enroll

Writing HTML into a Variable

Some of the report items, such as TITLE_LINES and SAS_LINES (described earlier) can hold a combination of line breaks, plain text, and HREFs. Simply inserting raw HTML into the variable did not produce the desired results. That is, coding such as the following did not produce a line break:

```
"<a href=…rest of HREF… </a>" ||
"<br>" ||
"<a href=…rest of next HREF… </a>"
```

After some trial and error, visits to the Online Doc, and SAS Tech Support, we arrived at the following:

```
ods escapechar='`';
"<a href=…rest of HREF… </a>" ||
`R"<br>"' ||
"<a href=…rest of next HREF… </a>"
```

The ODS statement tells SAS, in effect, "when you're writing HTML and see this character followed by an R, pass whatever you find in quotes directly to the HTML file – don't translate the
 into
." This results in a display such as:

sas
log
lst

The ODS statement does not need to physically precede the assignment statements using the escape charcacter. The only requirement is that it runs prior to the procedure that actually writes the HTML (in this case, PROC REPORT).

Changing Styles Within a Variable

The ability to emphasize comments was another problem whose solution lay in escape sequences. When it writes HTML, SAS normally includes font information such as name, style, weight, and size before every element of a table. This is why SAS-generated HTML files can become very large, and is one of many reasons to use Cascading Style Sheets (CSS).

If we want to change one or more attributes of an entire cell, there are several options available (traffic-lighting via user-written formats, for one). However, if the need is finer-grained, and the attributes need to change *within* the value of a table cell, we need an escape sequence. This tells SAS "I know you've been formatting this cell using all my carefully considered specifications, but let's override these for a moment. I'll let you know when I'm done." The general form of this is:

```
piece of char. string  ||
"`S={overrides}" ||
text with formatting overrides ||
"`S={ }"
text displayed with default formatting.
```

We added formatting to our comments as follows:

```
do i = 1 to 10;
   if comments(i) =: '!' then do;
      comments(i) = substr(comments(i), 2);
      special_start = '`S={foreground=red
                          font_weight=bold
                          font_size=2}';
      special_end   = '`S={ }';
      end;
      else do;
           special_start = ' ';
           special_end   = ' ';
           end;
   if comments(i)  ^= ' ' then
      title_lines = trim(title_lines)    ||
                    ' `R"<br>" '         ||
                    trim(special_start)  ||
                    trim(comments(i))    ||
                    trim(special_end);
end;
```

The technique is simple, and provides logical closure – by the time we're read to add a non-blank element to TITLE_LINES, we have assigned values to SPECIAL_START and SPECIAL_END. They may be blanks, but they *are* assigned and reassigned from one element of the COMMENTS array to the next.

A Note About PROC REPORT and HTML

Several concepts that are valid in "regular" output destinations such as plain ASCII files have no direct translation when writing HTML. The LINESIZE and PAGESIZE options, for example, have no meaning in the Web world. Likewise, DEFINE statement options in REPORT that are based on column size do not translate (nor is the lack of translation brought to your attention with a NOTE or WARNING in the SAS Log). The WIDTH and FLOW options in the DEFINE statement are ignored. The column width adjusts to the data being displayed, and text appears to flow automatically. The results are *usually* what you

would want to see.

On the plus side, character variables whose lengths would be impossibly wide in plain ASCII destinations become possible with HTML. The TITLE_LINES variable, for example, has a length of 2,000 bytes.

Version "Next"

At some point during any project, Version "n" has to be frozen, and good ideas have to wait until Version "n+1." Here are some "n+1" ideas:

- Maintain the meta data as a SAS data set. An observation would correspond to a program-output type (e.g., enroll – graph). This would eliminate the need to parse the ASCII meta data.
- The option for people who just can bear to throw things out: add an end of file line to the meta data. Anything beyond the line would not be read. This would allow "archival" reports and notes to remain in the program.
- Have an option to create pages for all types of reports in a single execution of the macro (e.g., TYPE =ALL). Currently, only one report type is processed per invocation.
- Enhance the output to show "families" of reports. Quite often, several reports are logically related, differing only by subsetting on demographic or other key variables. Report "n," for example, might be the entire treated population, report "n+1" might be the same report, but broken down by gender, and so on. A new meta data keyword could drive the identification of these groups on the HTML. The value of the keyword could be the name of the macro shared by the report family. The macro could be used in an HREF in much the same way as the SAS program.
- Provide an option to suppress printing of the "unexpected files" report. Similarly, provide an option to exclude Cx, or any other keyword group, from the output.
- A status indicator – "Final," "Draft," etc. could be added to the title. Given our directory structure, which has Prod and Test branches, we could derive the status directly from the directory name passed in the BASE parameter.

Closing Comments

The reports generated by this program are simple, easy to read, and not terribly difficult to code. The key to the program's effectiveness lies in output being stored and named consistently, meta data being entered and maintained correctly, and the program designers judiciously using HTML to produce clear and readable output. The meta data and the programs that utilize it are designed so that new keywords can be inserted into the meta data for new applications. Properly implemented, they should have no effect on the existing code base.

Contact Information

Frank DiIorio
CodeCrafters, Inc.
Frank@CodeCraftersInc.com

George DeMuth
Stat-Tech Services, LLC
GDeMuth@StatTechServices.com

Exhibit 1: Project Directory Structure

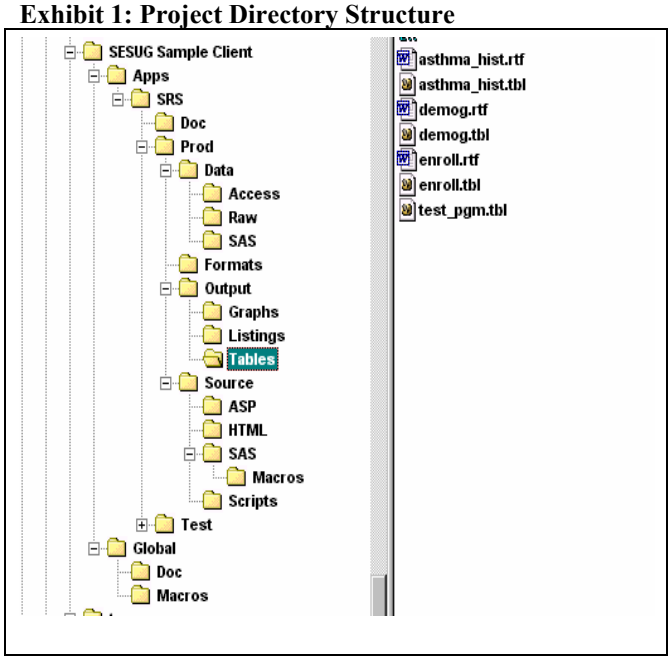
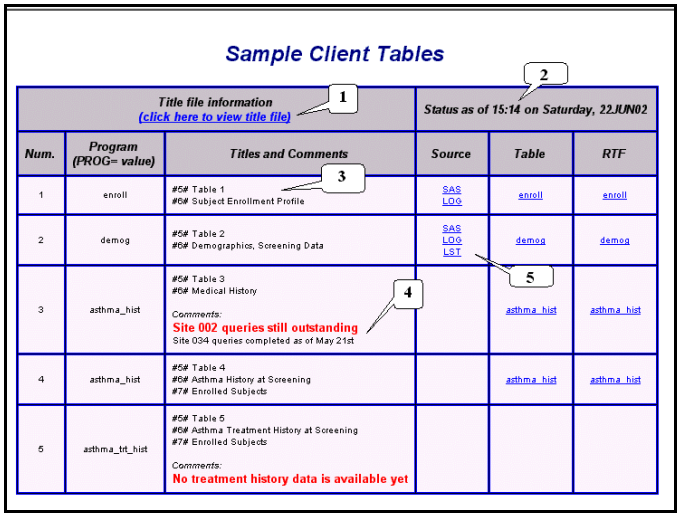


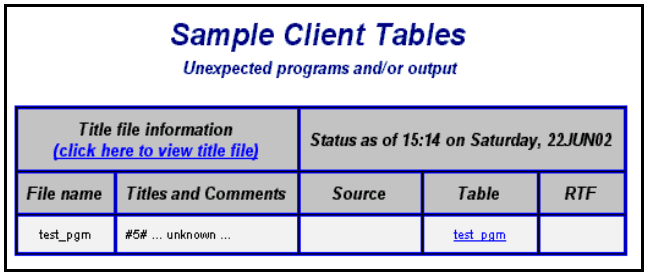
Exhibit 3: Sample Report



Sample Client Tables

Num.	Program (PROG= value)	Titles and Comments	Source	Table	RTF
		Title file information (click here to view title file) [1]		Status as of 15:14 on Saturday, 22JUN02 [2]	
1	enroll	#5# Table 1 #6# Subject Enrollment Profile [3]	SAS LOG	enroll	enroll
2	demog	#5# Table 2 #6# Demographics, Screening Data	SAS LOG LST	demog	demog
3	asthma_hist	#5# Table 3 #6# Medical History Comments: Site 002 queries still outstanding [4] Site 034 queries completed as of May 21st		asthma_hist [5]	asthma_hist
4	asthma_hist	#5# Table 4 #6# Asthma History at Screening #7# Enrolled Subjects		asthma_hist	asthma_hist
5	asthma_trt_hist	#5# Table 5 #6# Asthma Treatment History at Screening #7# Enrolled Subjects Comments: No treatment history data is available yet			

Exhibit 4: Reporting Anomalies



Sample Client Tables

Unexpected programs and/or output

Title file information (click here to view title file)		Status as of 15:14 on Saturday, 22JUN02		
File name	Titles and Comments	Source	Table	RTF
test_pgm	#5# ... unknown ...		test_pgm	