

A Reintroduction to ODS: The Philosophy of the Output Delivery System; *or,* How to Find Your Way Around All Those Features

by

Brian T. Schellenberger, SAS®

1. Introduction

When ODS was designed, there were certain scenarios that we had in mind and others that we never even considered. In addition, each feature and destination was designed primarily to solve certain classes of problems. While it is certainly *possible* to do many different things with the software, it is usually a lot easier if you can use it in the way it was designed to be used. Even if you must do something radically different, it's very helpful to understand what the paradigm is so that you have a better chance to work around it. Unfortunately, we in the ODS group have frequently not done a good job of conveying the "philosophy" of the ODS system, which makes it hard for you to work with it. This paper will attempt to correct some of that.

In addition, I will try to point out how the philosophy and history has a practical effect on your code, and some of the traps lying in wait for you, including those that arise because of perfectly-reasonable scenarios that we failed to foresee. The discussion will carry things forward into future releases of ODS.

2. Tables

The thing that most distinguishes ODS internally from what went before is not actually all the different forms of output, although this is surely what's most visible to the user, and the driving purpose of creating ODS in the first place; rather, it is the organization of data into *tables* of logical data, rather than mere collections of characters.

2.1. Prehistory

It's somewhat helpful to review where SAS was before ODS came along, how procedures worked then, and how they work now. This is particularly helpful in understanding the mysteries of the way that ODS interacts with DATA _NULL_ steps.

Originally, each and every SAS procedure did very much what you do in a v6-style DATA _NULL_ step that uses FILE PRINT: It printed a series of lines as ordinary text, choosing the column position (number of characters from the left) for each piece of text. For some time we had been aware that SAS had a reputation as a "back-end" system: people would often use SAS to do analysis and then take the results and translate them into a presentation form, often resorting to simply re-typing the output in order to get it into a form that would be acceptable when presenting reports.

And then this new-fangled "Mosaic" thingee, known as a "web browser", came along. It seemed to be rapidly gaining popularity, and it gave the effort to improve this situation more urgency. Although it could not, when the effort first started, actually format tables, the W3C committee already had a proposal for a table specification, and it seemed that Mosaic would soon be capable of presenting nicely-formatted tables. The first version of ODS (in SAS v7) supported only HTML as a new production output format (it also supported LISTING and OUTPUT, about which more anon), but the vast majority of the work for ODS was not the effort to implement the HTML driver; rather it was the work to convert *all* of the analytical procedures to a whole new way of doing things.

2.2. Data objects

Instead of just producing their output in an ad-hoc manner as they had done before, the procedure writers re-wrote their code so that it instead organizes the data into data *objects*, which are basically equivalent to SAS data sets represented in memory, and data *definitions*, which are defined using PROC TEMPLATE and control the layout of the formatted results. This has a couple of advantages. First, and most important, it allows the data to be represented as true tables in the various ODS output destinations. Secondly, it allows you, the user, to modify the presentation of the data.

As long as the procedures were producing simple lines of output, it was impossible to automatically render it correctly to different destinations, because it was impossible to tell what the various items represented. For example, if you had output like this:

```
1985 red Camaro $30,000
```

does that represent 2 columns (car, price) or 4 (year, color, model, price)? Without knowing that, you can't render it into a "proper" table with rules between columns. And more than that: you can't turn it into a SAS data set, which needs to know not only which data belongs in which column, but also the type and format that is associated with the data.

So the largest amount of work for ODS was the effort involved in having every procedure writer convert code that previously just wrote to the LISTING file to instead tag all of the data and describe it to ODS by creating a data object for the data and a definition for the appearance of the data (column order, formats, justification, headers, and so forth are all specified in the data definition, and therefore under full user control).

2.3. When data objects don't cut it

One issue that we did *not* consider to the extent that we probably should have was the situation with regard to DATA _NULL_ steps. It is a problem for ODS to handle DATA _NULL_ for exactly the same reason that was a problem for the procedures to do DATA _NULL_-like formatting: ODS needs to know something about the logical structure of the data, not just the appearance when rendered with a fixed-width font, in order to render it to a wide variety of output devices. The solution what we came up with is something that we call "batch mode," and it is actually used for a few exceptional procedures (such as PLOT, GRAPH, and CALENDAR) as well.

In batch mode, ODS acts very much as though it printed out the v6 output to the printer, and just pasted the result into the ODS output destination: a fixed-width font is used, and the actual lines which would otherwise be printed to the listing are captured and inserted into the current destination. This made it *possible* for ODS to handle any output, but it made some output decidedly inferior in appearance to other output. For procedures like PLOT this seemed like a reasonable tradeoff: we assume (and hope) that by now most users are using GPLOT rather than line-printer plots, and PROC PLOT is inherently graphical in nature, so there really is no other way to do it.

There were a few SAS procedures that also didn't fit very well into the table template model, primarily because of their data-driven nature, but for which the batch-mode solution is clearly not acceptable. For these procedures, known internally as the "gnarly four," we came up with another solution. These procedures make special calls directly into a special function interface that allows them to produce "real" tables with properly-formatted output but allows for greater flexibility than the strictly-rectangular format required of data objects. These procedures are: PRINT, REPORT, TABULATE and FREQ (n-way tables only).

Unfortunately, up through version 8.2, we have provided no way for users to get direct access to this more-flexible (but more verbose) approach. This will probably sound ridiculous to many of you reading this, and it sounds ridiculous to me now, but we thought at the time that once the user community saw how cool ODS was, we could largely wean them off of writing their reports "by hand" with DATA _NULL_ and get them to convert their reports to PROC REPORT and other reporting procedures.

For those who couldn't bring themselves to this, we even provided a DATA _NULL_ interface to ODS that allowed you write a dynamic data step that fed data to template. This does in fact allow you to write some reports that are quite different from the run-of-the-mill ODS procedure output, but it still has a fatal flaw: Because it is based on the table definition from the template, it is strictly rectangular--you can specify spanning columns, but that's it; once you get into the body of the table, it's a pure rectangular grid.

2.4. DATA _NULL_ plans for ODS

As a result, many users are still using old-fashioned DATA _NULL_ reports and failing to see many of the benefits of ODS. I am happy to report that we will *finally* be addressing this in our next release, which will provide a functional interface to the data step that will finally allow users access to the full functionality; this will allow you to produce tables, for example, like those that PROC TABULATE produces, which has heretofore been impossible. The new code will look roughly like this (very roughly--I'm just making up the syntax here):

```
data _null_;
  file print gendoc;
  table_start();
  row_start();
  format_cell("A spanning title",
             colspan=5);
  row_end();
  row_start();
  format_cell("multiple rows",
             rowspan=3,
             style=rowheader);
  :
  :
```

There will be a major drawback to this "new" way of doing things in the data step (aside from the verbosity): Reflecting the way that this interface works in the underlying code, this new interface will *only* work for producing output using the "formatted" ODS destinations. Reports produced in this way will not apply to the listing. This limitation was another reason we were reluctant until now to make it available.

2.5. Destination-independent input

One of the major goals of ODS is to allow you to produce output for numerous destinations from a single source without having to maintain separate source tracks for the various destinations. We try not to let this goal get in the way of letting users get their jobs done, but as much as possible we want to encourage portable solutions. Thus, we have the concept, for example, of the ODS style. If you were only producing HTML, you would probably find it easier to just use an existing style sheet, and we make it possible to do so, but if you use this approach then you will run into trouble as soon as you want to print the output, or produce a Microsoft® Word document from it.

We have, recently, however, made some "breaks" from the considerations of this philosophy. For numerous technical reasons, mainly having to do with the internal architecture, it is impossible to process the in-line formatting feature in the LISTING, or even to skip over the formatting codes. So if you use this feature, your listing gets completely messed up. Nonetheless, we chose to ship this feature in v8.2. Similarly, the new flexible table interface will not support the LISTING or OUTPUT destinations; nonetheless, we are planning to ship it in v9. So we are now loosening our philosophy here a bit, but there has to be a compelling benefit before we add a feature to the data objects or definitions that will result in damaged or lost output for some of the destinations.

Note that this is quite different from the case where one destination can support a feature and the others can't, but they can quietly ignore the request. *That* sort of thing, in fact, happens all the time; indeed, it's fundamental to the idea of the destination-independent input; otherwise, we'd be able to only support the features of the least common denominator, and users would be forced to insert raw HTML or RTF into their input if they wanted special features of those destinations. As it is, customers are sometimes forced to do this anyway, but we try to minimize the necessity to do this; when it is done, it is much more difficult to move your report to another output format, and we on the ODS teams are in the business of trying to make that easier instead.

3. The 'special' destinations

Each of the different ODS destinations was designed with a different purpose in mind, and although it is possible to use the destination for some other purpose, it is easier (if management and government regulations make it possible) to use the destination suited to your intended purpose. Most of the destinations are designed to produce formatted output, but there are three "special" destinations that don't quite fit in with the others.

3.1. The LISTING destination

The first destination, of course, was the LISTING, for the listing is essentially the equivalent of the pre-ODS v6 output. However, it's not really the *same* as the v6 output, because even when you get output in the LISTING destination, it's still being turned into a data object and it's still using a table template to control the output. Thus ODS is really being used all the time, even when you're not "using ODS." Like everything else, there is an exception, of course: the "gnarly four" (PRINT, REPORT, TABULATE, and FREQ n-way tables) actually are executing their "legacy" v6 code when producing listing output.

But for other procedures, if you change something via the table template, those changes will affect the LISTING as well as the new ODS destinations. In addition, the folks responsible for the analytical procedures in SAS took advantage of the fact that the procedures were being reworked and that labels and such were controlled by templates to make the SAS listing output much more consistent than it was before. If you have two different procedures producing an ANOVA table, for example, they will produce it in the same way; indeed, they will use the same template to describe the table; it will be referenced by the templates for each of the procedures.

The purpose of the listing destination, as far as the ODS team is concerned, is simply to allow those people who want to do so to go on as they always have. Because it is intended purely for "legacy support," we do as little as possible with it, and new features don't necessarily apply to the listing.

3.2. The OUTPUT destination

The OUTPUT destination is a rather special destination: it produces SAS output data sets. Its purpose is to make PROC PRINTTO a thing of the past. Because ODS already knows the logical structure of the data, including having the data available in its native form, ODS can output a data set that represents exactly what the procedure has to work with internally. This means that you don't have to parse through the output and figure out what sort of informat will properly reverse the formatting that the procedure did and that sort of thing. The data sets, of course, are intended to be used for further analysis or for particularly sophisticated reports, where perhaps you want to combine similar statistics across different data sets into a single table. SAS analytical procedures can generally (there's probably some exception that I don't know about) only process one data set per run, and this is, among other things, a way around that limitation.

3.3. The DOCUMENT destination

The DOCUMENT is the newest "special" destination; in fact, it's not actually out in production form yet; that's coming in v9. However it *is* already available experimentally. The DOCUMENT destination does for your entire output stream more or less what OUTPUT does for individual data sets: it makes it available in "raw" form, suitable for further manipulation. However, in this case, it's not even transformed into a SAS data set, it's kept in the original internal representation as a data object + template. Once the output is in a DOCUMENT form, it is possible to re-arrange it, to duplicate or remove certain tables or the output of entire procedures, and then to generate output for one or more ODS output destinations using the newly-transformed output tree. Thus, DOCUMENT allows you to transform your report as much as you like without having to re-run your analysis or re-query your database.

Unlike other destinations, the DOCUMENT has a substantial GUI interface, and it is our expectation that this will be the primary interface for most users of the DOCUMENT destination. However, everything that is doable through the GUI is also doable through batch commands (ODS DOCUMENT and PROC DOCUMENT), so it can be used even in a batch environment.

4. The 'formatted' destinations

The remaining ODS destinations are what I sometimes call, for lack of a better term, the "formatted" destinations. These are, basically, the destinations to which styles apply: the destinations that support concepts like "font" and "color." (For the most part--again there are exceptions.) These constitute what we generally *mean* when we speak of "ODS output" even though LISTING and OUTPUT are in fact ODS destinations as well.

All the aspects that control the appearance of the formatted destinations beyond what the LISTING can do are controlled by two mechanisms: options on the ODS statement for the destination and the ODS style mechanism. The options on the ODS statement control three sorts of things:

1. Attributes that are extremely specific to a given destination, such as stylesheets for HTML.
2. Attributes that are global to the document, such as AUTHOR and table of contents generation.
3. Attributes that we expect users to change on virtually every document, such as the output file name.

The style attributes, on the other hand, control the way that individual elements are rendered. This can be confusing, since the document has an overall style, specified on the ODS statement, but that overall style is simply a (hopefully coherent) collection of style elements each of which lists the style attribute values for different parts of the report.

The style attributes are intended to prevent the necessity to insert destination-specific coding (such as raw HTML) into the document by providing a mechanism to describe what the document is intended to do which each output destination will interpret in the most reasonable way possible. Because not all destinations are the same, not all attributes can be interpreted by all destinations. The style is deliberately defined so that any aspects of the style that cannot be handled by a given destination are ignored by it. For example, PostScript does not support active links, so the URL= attribute is ignored when producing PostScript output.

4.1. HTML

The first formatted destination that was shipped, and the only one that was production in version 7, was ODS HTML. The legacy of this history is immediately evident if you look at the design of the style attributes: many of them are based directly on the way that HTML, and in particular Internet Explorer, handled attribute specifications. (We based our code on Internet Explorer because, whatever else one might have thought of the browser wars, the formatting control for table attributes was unquestionably superior in Internet Explorer.) Another fateful decision we made was to *not* assume that users would have version 4 browsers available; indeed, we still ship HTML so that by default it produces HTML3-compatible output rather than HTML4 (stylesheet) output, although we have made the option to produce stylesheet-based output available since at least v8.0.

HTML was the first formatted destination we shipped because it was a format which a lot of users wanted us to support and it was quite different from what we were doing before. I believe that it was even thought at one time that LISTING, OUTPUT, and HTML might be sufficient--that perhaps that was all that we would need for ODS to do. However, as internal development proceeded, it soon became clear that it would not long suffice: browsers, frankly, do a terrible job at printing, especially of long or wide tables (of which SAS has quite a few). For example, tables headers don't repeat when you cross pages. Worse than that, if the table is too wide, the rightmost columns are simply dropped--they just vanish. And HTML doesn't provide any way for the programmer to provide "hints" about how to print it, either. Plus the fonts (using HTML3) are not of a predictable size so even if we wanted to break up the tables ourselves to avoid printing anomalies we wouldn't know where to break them.

ODS HTML is thus intended only for on-line use, not for printing. We do sometimes still get queries or "bug reports" about problems with printing HTML, and we explain that you can't do that; that's what ODS PRINTER is for . . .

4.2. PRINTER & friends

Because ODS HTML was suitable only for on-line (non-printing) use, we introduced ODS PRINTER as a production destination in v8.0. The ODS PRINTER code is actually a sort of "meta-destination" that covers both printing to physical printers (any Windows printer under Windows; PCL and PostScript printers on other operating systems) and producing portable PostScript, PCL, and PDF files. (The ability to produce PDF files is new in v8.2.)

What all of these have in common is that they are all Page Description Languages: they describe precise positions on the page to place each line of text, each rule, and each graphical element. These formats are not, in general, editable or alterable by the user; thus, the output from ODS PRINTER is intended to be the final form of the report, with everything placed carefully so as to be aesthetically pleasing, and with care taken to deal with overall and overwide tables in a reasonable manner.

4.3. ODS RTF

ODS RTF, introduced as a production destination in v8.1, is the destination that seems to cause the most confusion over philosophy and the main inspiration, in fact, for this paper. One thing that we should make clear right away (and I believe that our documentation and talks do make this pretty clear) is that ODS RTF exists in order to produce output for Microsoft Word. There are other applications that can read RTF files, and ODS RTF might happen to work with some of them, but it is not designed to, so if you run into problems that are specific to other applications, you are more or less on your own. (You can report any such problems, but they will be treated as low priority since we never claimed that it would work elsewhere.)

But the bigger source of confusion is the issue of "vertical measurement." When we designed ODS RTF, we did not design it primarily to *present* information in a final form; after all, ODS PRINTER already does a nice job of that. Instead, it is designed to allow you to *edit* the file, perhaps to add additional notations, or to create a report by typing the report "around" the SAS output containing the data analysis for your report. For this reason, we don't do "vertical measurement." In other words, we don't figure out the optimal place to position each item on the page. Unfortunately, Word demands to know the widths of table columns, and it doesn't know how to "panel" tables if they are too wide for the page, so we measure the text and the tables to see how *wide* they are (horizontal measurement), so that we can set all the column widths properly and, if necessary, divide the table into panels, or strips, if it is too wide to fit on a single page.

But we deliberately avoid doing the sort of measurement that ODS PRINTER does in the vertical direction. This leads to some output that is less than optimal; for example, when rendering over-wide output from PROC PRINT, ODS PRINTER can render it like this:

Whereas ODS RTF may wind up with the pages unbalanced, like this:

In the case of ODS PRINTER, each page (or column, in the example output, which is actually using two columns) has the top half filled with the the "left" half of the table and the second half filled with the "right" half. The effect is to keep all of the values for a single observation in close proximity. In ODS RTF, on the other hand, the *entire* "left" half of the table is output and then the entire "right" half is rendered with total disregard for the page breaks, because RTF doesn't know where the page breaks will occur anyway. As a result, values from a single observation can wind up being quite far away from each other.

But the problem is that if we measured in the vertical, then our calculations would be wrong as soon as the user started altering the text. For example, if they inserted an explanatory paragraph before the start of table, and we had put forced page breaks into the table (as we do in ODS PRINTER), then instead of breaking optimally, the table would instead have page breaks at all the *wrong* places, and the first page of the table, for example, would end and Word would break it to the second page (as it does when it runs out of room on one page). Then just a few rows later we would hit our forced page break and leave the second page almost empty. To avoid this sort of difficulty, what we do instead is to set up table headers using the RTF mechanism to mark them as such, and just let Word do page breaks whenever it needs to. Word is smart enough to repeat table headers and titles on its own as long as they are properly marked in the RTF (as SAS output is). Since it knows how much room there is on the page even when the file has been edited, this always yields consistent results.

From time to time we get calls from people about one or the other of the glitches that arise because of the lack of vertical measurement in RTF, and we try to explain that they should use ODS PRINTER for output *presentation*; RTF is designed for *editability*. And this does make sense; however, the factor that we had not counted on is that some companies actually *do* use Word as an exchange/presentation format, attaching Word documents to e-mail for example. This is a particular problem for the pharmaceutical industry, because RTF is one of the formats in which the FDA will accept electronic submissions. (Intense interest from pharmaceutical customers was a major impetus behind the implementation of the ODS RTF destination in the first place.) Now, this makes little sense to me personally; I should think that one would want to encourage formats for data presentation that make it difficult rather than easy to manipulate the reports after they were finalized, but it's a reality nonetheless.

For this reason, we now seriously considering breaking RTF up into two different destinations: One that does full measurement, treating RTF like another page description language and invoked, perhaps, as ODS PRINTER RTF, and the other treating RTF like we do now as an editable destination. This, however, would be in a release later than v9 as it is a substantial project and we have only very recently become convinced that we must seriously consider undertaking such a project.

4.4. ODS MARKUP

The final formatted ODS destination, which will go production with v9, although it is already available experimentally in v8.2, is ODS MARKUP. This destination was newly implemented in v8.2, but it has its roots in the experimental XML destination in v8.1.

The primary motivation for creating the abortive ODS XML destination was, of course, to produce XML, which is the up-and-coming standard for internet (and intranet) data and report interchange. The problem that we ran into with ODS XML is that XML is not a traditional markup language in which a given language element means some specific thing; rather it is a meta-language, used to create other, more specialized languages. (If you are familiar with SGML, XML is basically a subset of that.) The result is that lots of different people want to get XML output, but everybody wants their *flavor* of XML. Originally we planned to handle this somewhat like we did with ODS PRINTER: Initially we'd release one or two common varieties of XML and then we'd gradually add to the varieties we support over time, releasing one or two new flavors in each release.

It quickly became apparent, however, that whereas new page description languages are enormous undertakings and a significant new one is invented perhaps every 5 years at most, far longer than our release cycles, XML makes it so easy to create new varieties of XML that various standards committees, industry groups, and even individual companies are inventing new varieties of XML that they'd like us to produce on practically a weekly basis. Clearly the ODS PRINTER "output driver" model was not a viable approach.

So Eric Gebhart, who was responsible for XML (as well as HTML), came up with the idea of "tagsets." A tagset is a new sort of definition that you can create in PROC TEMPLATE. Just as table templates (and their associated sub-types) describe how to lay out a table such that the user can modify it, and style templates describe the style of the output such that the user can modify it, so tagsets describe how to produce a markup language output such that the user can modify it. Each variety of XML can be specified as a new tagset, and while we will ship with a fairly reasonable collection of XML tagsets to allow SAS to produce a fair variety of XML, the more important thing is that you as a user need not wait for us to support some new variety of XML; you can go ahead and implement it yourself.

Moreover, HTML is itself a valid XML language, so HTML can itself be specified via the tagset mechanism. Although the HTML that you get with ODS HTML will continue to use the existing driver through v9, there are a number of *HTML* flavors available in the tagsets for v9 (which you can play with experimentally in v8.2 as well). In v10 (or whatever we wind up calling the release after v9), we plan to make the the primary ODS HTML use the HTML4 tagset. (The existing ODS HTML driver will still be available as the "legacy" driver via ODS HTML3.) Using the tagsets, we will be shipping HTML4 (a modern css-based HTML), CHTML (compact HTML for hand-helds) IMODE (another standard for handhelds, WML (Wireless Markup Language), and others.

And the tagset mechanism is flexible that it isn't even limited to XML at all; there are also tagsets to implement CSV (command format for spreadsheets), L^AT_EX (replacing our previous C-based implementation of an always-experimental destination), and even troff (traditional Unix markup format). I even wrote a tagset once that would allow you to produce SAS data step *code* from a dataset. It looks like this:

```
proc template;
define tagset Tagsets.datastep;
  notes "This is the Datastep definition";
  define event table;
    start:
      put "data;" NL;
    finish:
      put "run;" NL;
  end;
  define event row;
    finish:
      put NL;
  end;
  define event table_head;
    start:
```

```

put "input ";
  finish:
    put ";";
end;
define event table_body;
  start:
    put "cards;" NL;
end;
define event header;
  start:
    trigger data;
  finish:
    trigger data;
end;
define event data;
  start:
    put " " VALUE;
end;
end;
run;

```

As ODS "renders" the data set, it triggers events, and the code above describes what output to produce for each event. To use it, you just do a PROC PRINT of the data set:

```

ods markup type=datastep file="b_out.sas";
proc print data=sashelp.class; run;
ods markup close;

```

And you get this result:

```

data;
input  Obs Name Sex Age Height Weight
;cards;
  1 Alfred M 14 69.0 112.5
  2 Alice F 13 56.5 84.0
  3 Barbara F 13 65.3 98.0
  4 Carol F 14 62.8 102.5
  5 Henry M 14 63.5 102.5
  6 James M 12 57.3 83.0
  7 Jane F 12 59.8 84.5
  8 Janet F 15 62.5 112.5
  9 Jeffrey M 13 62.5 84.0
 10 John M 12 59.0 99.5
 11 Joyce F 11 51.3 50.5
 12 Judy F 14 64.3 90.0
 13 Louise F 12 56.3 77.0
 14 Mary F 15 66.5 112.0
 15 Philip M 16 72.0 150.0
 16 Robert M 12 64.8 128.0
 17 Ronald M 15 67.0 133.0
 18 Thomas M 11 57.5 85.0
 19 William M 15 66.5 112.0
run;

```

Because of the immense flexibility of the tagsets, the name ODS XML clearly didn't suffice to describe it anymore so it is now called ODS MARKUP. The name ODS XML can still be used; it invokes ODS MARKUP to produce XML. We have similar aliases set up for a number of varieties of markup, so that you can produce, for example, troff using ODS TROFF. You can even do this with user tagsets: I could have said ODS TAGSETS.DATASTEP above rather than ODS MARKUP TYPE=DATASTEP.

The MARKUP destination is so flexible that it has already replaced ODS LATEX and in the future it will replace ODS HTML as well. However, in anything like its current form it cannot replace ODS PRINTER or ODS RTF, because it does have one major limitation: it cannot do text measurement; therefore, it cannot produce output for a page description language or a hybrid language like RTF which require all of the text to be measured and placed at a specific position on the page (in the case of RTF, columns must have calculated widths). So it can only be used for *markup languages*; that is, languages in which no locations are required, either because the language isn't meant for printed output at all (like XML and CSV) or because the language goes to a formatter that does the positioning (like HTML and LATEX).

5. Conclusion

The Output Delivery System was originally designed to optimize output from SAS procedures, which by and large process ordinary rectangular tables. The impact of this legacy can still be felt in the difficulty of doing free-form (non-rectangular) reports and/or forms using ODS, but we are working on this.

The destinations are optimized for different purposes:

LISTING	is a legacy destination to support people who aren't interested in new features.
OUTPUT	was created to eliminate parsing PROC PRINTTO output.
DOCUMENT	will allow you to produce multiple reports with a single run of the analysis.
HTML	is only meant for on-line viewing.
PRINTER	is for presentation-ready printable reports.
RTF	is designed to allow editing, not optimized for final output
MARKUP	is meant to allow ultimate flexibility for markup languages.

5.1. Legal niceties

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 registered trademarks or trademarks of their respective companies.