ODS Markup: Tagsets by Example

September 22, 2003

Preface

The SAS Output Delivery System's (ODS) markup destination appears to be much like all the other destination types that ODS is capable of. In reality it is quite different.

The other ODS destinations have a fixed output type. The RTF destination can only create RTF output. The printer destination can create postscript, PCL, and PDF. But ODS Markup can create any number of output types, all of which can be created or modified by anyone.

ODS Markup is able to do this because it is really a framework which uses a tagset to define what it should print to it's files. The other destinations, including the old HTML destination were hard coded and compiled as a part of the SAS executable.

A tagset can be described as a group of events or functions which use a programming syntax similar to other programming languages such as perl, python, shell, and datastep.

We can create and use as many different tagsets as we like. There are a number of tagsets that come standard as a part of SAS.

Because there are so many different tagsets, ODS Markup is rather chameleon-like. ODS HTML is really ODS Markup. So is the CSV destination, and CHTML, excelXP, LaTeX, Troff, etc. The only difference between these different ODS destinations is the tagset that is in use. They are all really ODS Markup, we just don't call it that.

When we use ODS Markup we are using a tagset. The tagset determines the type of output. Therefore, a tagset defines an output destination! This realization has far reaching implications. It means that we can change the html that ODS HTML generates. It means we can create a new HTML destination with your corporate style and headings. We can create a new XML destination to allow data interchange with a business partner or client. We can even create a simple flat file format that can simplify processing by other programs. The possiblities are endless.

It stands to reason, that if we want to use ODS Markup to our best advantage, what we really want to do is use tagsets. That is what this book is going to explore. What are these tagsets and how do we use them? You will find that tagsets can simplify otherwise complex problems in a way that allows reuse and flexibility that would not be possible with out them.

ableofcontents

Part I

Introduction

Chapter 1

Introduction

ODS markup and tagsets are the best way to create markup output from SAS. This chapter will talk about some of the reasons for learning tagsets. As well as what they are and the history behind them.

1.1 Why Learn Tagsets?

There are several tagsets provided by SAS for your use. Everything from HTML4, Compact HTML, CSV, to Troff and LaTeX. So why would you want to learn how they work? Or how to program them? My answer is to ask you several questions.

1.2 What are Tagsets?

Tagsets are templates, a program, compiled by proc template. But tagsets were created to simplify th creation of GML [1] output. Because of what they do, Tagsets are very different from any of the other template languages you may already be familiar with. This chapter will give a short history of how and why tagsets came about. This should help give a broader view and therefore, a better understanding of how tagsets work. The second part of this chapter will explain the basics of using proc template with tagsets.

1.3 A short History of Tagsets

ODS Markup has it's roots in ODS HTML. I created ODS HTML in 1995. The internal model for ODS HTML was created while keeping other markup languages in mind, troff and LaTeX in particular. The problem with ODS HTML is that it was fixed, the output only came out one way. The HTML it created worked well enough most of the time. But it could not be everything to everyone. The HTML could only change with each release of SAS.

[1] General Markup Language

1.4 Markup Languages

1.4.1 In the Beginning There Was Roff

1.4.2 Then there was LaTeX

1.4.3 SGML and friends

1.4.4 Other Formats

1.4.5 What it all means

Chapter 2

The basics

This chapter will cover the basics of using ODS Markup and the Template Procedure. ODS markup relies upon tagsets to define how it should work. Indicating which tagset to use is the only thing that differentiates the ODS Markup statement from any other ODS destination. A tagset is defined using the template procedure. The amount of template knowledge needed is not that great, but it is good to know the basics of using it.

2.1 Tagsets and Proc Template

A tagset is a template. That means it is a proc template job that runs in SAS. The Template Procedure is capable of processing several different types of templates. All of them are quite different from each other. The thing they all have in common is the basic inner-workings of proc template. When a template is run, the procedure compiles it into a binary format which is then stored on disk in an itemstore. Typically this file will be in sasuser. But can be placed anywhere using the 'ODS path' statement. If administrative priviledges are available the template can be put in sashelp where it will be available to all users on the system. Once a template is compiled there is no need to compile it again unless it's itemstore is deleted.

2.1.1 The tagset directory

2.2 Using a tagset

To use a tagset, all we need to do formulate a proper ods statement. ODS Markup is not all that different from ODS HTML, in fact, ODS Markup can do everything ODS HTML can, plus a little more. The simplest form of the ODS Markup statement is shown in Figure **??** on page **??**.

Chapter 3

Tagsets: how do they work?

Tagsets are a hybrid of procedural and non-procedural, declaritive programming languages. A tagset is a collection of event definitions. Each definition can have procedural elements, but the Events themselves are not procedural. That's why they are called events. Events do happen in some order, but with variation. They are like the events in your day. The alarm clock, brushing your teeth, eating breakfast, going to work, etc. They do have some predictability, but are not exactly the same all the time. This chapter will use a simple plain text tagset to show how tagsets work. As the tagset evolves the output created will reveal the fundamentals of tagset behavior.

3.1 Data and context in time

3.2 Event Requests

A tagset defines what is to be done with the data for any given event. But it is not necessary to define all possible events. Only events which are meaningful to the output we are trying to create need to be defined. For this reason, it is more accurate to think of these data bundles as event requests. You may have a title statement in your SAS job. When it comes time, ODS will bundle up all the data, and metadata which defines that title. The title event will then be requested. If the tagset in use has a definition for the title event then something will happen. Otherwise, nothing will happen.

3.2.1 A few variables

3.3 Our First tagset

It's best not to get bogged down in the details, or to spend much time worrying about how all this works. Tagsets are self revealing, the best way to learn them is to play with them. So we are going to start with something simple. It will all make sense in time.

3.3.1 Define statement

3.3.2 The data event

3.3.3 The header event

3.4 Fleshing out the plain_text tagset

This is all very nice but to create something more useful we are going to need some more events. The most basic events are fairly intuitive. All tables have a handful of basic events. We've seen the data and header events. The other basic events are, table, row, head, body, and foot.

3.4.1 head, body, and foot

3.4.2 Titles

Chapter 4

Modifying Existing Tagsets

While you may find yourself creating a tagset from scratch, It is far more likely that all you'll want to do is modify the behavior of a tagset you already have. Modifying a tagset is fairly easy to do because tagsets can inherit events from each other. It's also much easier to identify which events need changing. In this chapter we will introduce a few new concepts while showing the steps needed to modify a tagset through inheritance.

4.1 Changing the delimiter for CSV files

4.1.1 Finding the Events

4.1.2 Simple If's

4.2 Making the changes

Even with the new syntax we just discussed, our job is pretty easy. All we really want to do is those comma's to semi-colons.

4.3 A better CSV tagset

Our new tagset is nice, but it's not very flexible. It's not that hard to write a new one if we want a new delimiter, but wouldn't it be nice to have a tagset that would use any delimiter we gave it?

4.3.1 Macro variables

4.3.2 The Initialize Event

4.3.3 The set statement

4.3.4 The New CSV Tagset

4.3.5 Tagset Alias

4.4 Summary

This chapter introduced Several new features which are used extensively in all tagsets. Inheritance is the primary way of modifying and creating new tagsets. Simple If statements are the most common form of controlling the program flow within events. Macro variables and the tagset alias are ways to affect the way a tagset works from one usage to the next.

Chapter 5

The Path to Enlightenment

So far all of the examples have given away the events and variables needed. But the ability to find events, and their variables is required to be truly effective at programming tagsets. This chapter will explain several techniques that will allow you to understand how tagsets work, which events you may want to use, and the variables that are bundled in those events. This where you learn to get tagsets to reveal themselves.

5.1 Finding Events

So far, to keep things simple, we given a few event names away. But there are many more events than those few we have revealed. The events also vary according to what you run and which output file you look at. The events for the body file will be different than the contents file which will be different from the stylesheet file.

5.1.1 The Short_map Tagset

5.2 Anatomy of the Short Map Tagset

So how do we know that we are seeing all the events? The answer is because these tagsets use the default event. Normally, if an event is not defined, the event request is quietly forgotten. But a tagset can have an event that is designated the default. If an event is not found, in a tagset that has a default event, the default event is triggered instead. The code for the shortmap tagset can be seen in figure **??** on page **??**.

5.2.1 The Events

5.2.2 The Default Event

5.3 The Tagset Attributes

The first thing we see is notes. This is proc template's version of a comment. Notes are reprinted when the tagset is sourced from within proc template.

5.3.1 Map and MapSub

5.3.2 nobreakspace

5.3.3 split

5.3.4 indent

5.3.5 Stacked Columns

5.3.6 Image Type

5.3.7 Output Type

5.3.8 summary

5.4 Finding Variables

In addition to knowing when events occur it's also important to know what data is available for any given event. The mapping tagsets show some of this but there are far too many variables to print them all the time.

5.5 The Putlog Statement

This chapter has shown you how to reveal the inner workings of tagsets. But sometimes we need just a little more. There is another statement that is quite handy for quickly showing what is going on. Putlog is like put except that instead of printing to the output file it prints to the SAS log.

5.6 Partial Enlightenment

We've got some powerful tools which can help us find the best way to write or change any tagset. When starting from scratch one of the mapping tagsets is a good place to start. They are also good for comparing against existing tagsets.

Part II

Technicalities

Chapter 6

Creating Variables

We’ve used a set statement to create our delimiter variable in the CSV tagset. Before continuing it would be good to explain the different types of variables and the methods of creating them. This chapter is dedicated to explaining the different types of variables that can be created and used in tagsets.

6.1 String Variables

The most common variables are string variables. Any variable created with the set statement will be a string. Tagset variables are variable length, meaning that they are always the size of the text they contain. If a variable is assigned an empty string then it really doesn’t exist at all. But that’s ok, because referencing a non-existant variable is allowed, in fact it’s the easiest way to create conditional values. Setting a variable to ’true’ and unsetting it for false is the most efficient way to use a variable of this sort.

6.2 Lists

Set can also create variables that are lists. Lists are denoted by a suffix of ’[]’. If the brackets are empty the set adds a new entry to the end of the list. Lists are 1 based, the first entry always has an index value of 1. If there is a number within the brackets the value at that location in the list will be replaced with the new value. If the list is not that long, then the value is appended to the end of the list. If a list variable is referenced without a subscript the return value is the number of entries in the list.

6.3 Dictionaries

6.4 Numeric Variables

Numeric variables can be quite useful for list manipulation and many other things. Numeric variables can be created with the eval statement. The eval statement has two arguments. The first is the variable to set. The second is a where clause. Eval is one of the few statements that cannot have an if condition. The type of the variable created with the eval statement depends on the where clause. If the result of the clause is numeric then the variable will be numeric, otherwise it will be a string. One advantage of where clauses is that functions can be nested. But a set statement is much more efficient, so if a string is desired set is the better choice.

6.5 Stream Variables

Occasionally the output we wish to create does not fit the event model used by tagsets. This is where stream variables are most useful. Stream variables are more like a temporary file than a variable. The set and unset statements work with stream variables, but it is much more common to create or append to a stream with the open statement. Once a stream is open all output will be directed to the stream. Streams automatically compress anything that is written to them. As long as the output is small the stream will stay in memory. As it grows it will be written to disk. A stream is closed with a close statement. Only one stream can be open at a time, so opening a stream will close any stream that was previously open. When using a stream variable in a put or set the variable is prefixed with a $$.

6.5.1 Stream Specific Statements

6.6 The Putvars Statment

Putvars is a special statement that allows us to see all the values for a given category of variables. The first argument to putvars is the category we wish to examine. The categories are shown in table **??** on page **??**.

6.7 Bringing it all together

It's quite easy to create a tagset to test all of these things. We can use the initialize event, or in SAS 9.0 or earlier we can use the doc event. The code example on page **??** combines the examples in this chapter into one event. The output is shown in **??** on page **??**.

Chapter 7

Procedural controls

We've already been introduced to Tagset's if syntax. With SAS 8.2 and 9.0 if statements were kept to a few simple choices. This simple if statement is still the most efficient and most frequently used type of conditional programming that tagsets have. This chapter will explain how those if statements work.

7.1 Simple If's

An if can be placed at the end of almost any tagset statement[1] . But it must be preceded by a /. Usually an if follows the /. But that is optional. In fact, the keywords 'when' and 'where' are also allowed. But it's all personal preference since none of the words mean anything. There are two cases where the keyword following the / is important. Breakif and While cause the if to do more than a simple test.

7.1.1 Built in tests

7.2 Where Clauses

We've already introduced the eval statement which uses SAS where processing to do math and other operations. In addition to the simple if conditions above we can also use where clauses. It turns out that they are usually used when doing numeric comparisons, but some problems reguire more complex if's. The where processing provides that.

7.3 Break

Break is a statement just like put or set. When encountered the program flow exits the event immediately. This is useful for skipping the end of an event when the remaining statements are not desired. Here is an example based on the initialize event from the example in in figure **??** on page **??**.

[1]eval and done are the only statements that cannot have an if

7.4 Breakif

Breakif is one of the two special keywords that can be used to define an if test an a tagset statement. When looking through the tagset code, there are many times when it is desirable to do something and then break if a certain condition is met. With breakif it is possible to write this with one line instead of two. That means one test evaluation instead of two. Breakif[2] can be quite useful in these situations because it combines break with the execution of another command based on an if. Using our same example we can do this:

7.5 Do blocks

Do blocks[3] provide a way to group several event statements together under one condition. do blocks start with a Do statement, end with a Done statement and may have any number of else statements in between. Using the same example we can now do this:

7.6 Do While Loops

Using the do statement it is also possible to create loops. We can do this with the help of another special keyword. Like breakif, while[4], has special meaning. it causes the do block to loop as long as the while test is true. A Do /while block can also have an else, or even multiple else /if's. If the initial while test is false, execution will fall to the else. But if the while block executes at all then the else will not be executed.

7.7 Iterating through Dictionaries

Since we have looping, iterating through a dictionary of values would be a very nice thing to do. Looping through a list is easy, all we need is a counter. But dictionary's are not so easy. To loop through a dictionary we have the iterate and next statements. Iterate and next work similarly to putvars in that they populate the special variables _name_ and _value_. Iterate sets up the iterator for us and populates the first name and value into _name_ and _value_ respectively. From then on we only need to use the next statement. Of course an iterator will work on a list too, but there won't be a _name_. These examples are awfully simple, putvars could have done this in one line. The output can be seen in figure **??** on page **??**.

7.8 Bringing it all together

The code example on page **??** shows everything covered in this chapter in one event so that it can be easy to see how things work.

[2]Available in SAS 9.1 and later
[3]Available in SAS 9.1 and later
[4]The while keyword is only valid on the Do statement

Chapter 8

Trigger Happy

ODS triggers, or requests, many events internally. As we have seen, it is also possible to trigger events from other events. This chapter is going to explain the trigger statement and how to use it.

8.1 Simple Triggers

We've already seen trigger used for simple cases where the triggered event is a simple event without a start or finish. This type of use is very straight forward and does exactly what we would expect. In the following example the message from the do_stuff event will be printed twice.

8.2 Events with a state

Events can have what is called a state. In the example above, the doc event has two states. The first time it is triggered it's state is start. The second time it is in the finish state. The do_stuff event is stateless. It does the same thing always. The 'state' variable indicates the current state. It's value will be either start or finish.

8.3 summary

Triggered events are great way to compartmentalize commonly used code. When combined with if and breakif they are also a good way to control the program flow. If your events are getting complicated with lots of do / elses then it's probably time to create some new events and trigger them. Triggered events are just as fast as inline code, so performance is not an issue. Readability and maintenance of your tagset is.

Part III

Intermediate Examples

Chapter 9

Styles and Tagsets, A perfect match

Understanding how to use styles with tagsets can simplify many problems. Styles can help make a tagset more versatile and reusable. They can also help to make the output less verbose and more compartmentalized. In this chapter we will demonstrate the basics of using styles with tagsets. The best way to start is with a simple tagset designed specifically to show how styles work. From there we can move on to actual problems and their solutions.

9.1 Starting Simple

The easiest way to examine any problem is to break it down as much as possible. We can learn how styles work with a tagset that has one event. We can choose any event. The table event is a good place to start. We already know about it, and tables have some interesting style attributes.

9.1.1 Embedded styles

9.2 Styles of your own choosing

So far we have been using the style names that ODS decided to use. But it is possible to create and use style elements using the names you desire. We can do this by using the the event's style attribute. The code in figure **??** on page **??**. Adds a new style element, darkbox, and then uses it in the table event. The output in figure **??** on page **??** shows two style class events and one Table event which reveals our new style. Note that any new style elements in our ODS style result in another triggering of the style_class event. This is a very handy behavior that we will put to good use.

9.3 Getting the whole style

Having a separate place for style definitions is a good thing. But sometimes it is necessary to get all the style information all of the time. There is another event attribute which allows this to happen. If pure_style is set to yes, and the style attribute has been set, then all attributes of a style will show up in the event. If this behavior is desired for all events the pure_style attribute can be set as a tagset attribute instead of an event attribute. In example **??** on page **??** pure_style is set to yes. The resulting changes to the output from the table event can be seen in figure **??** on page **??**.

9.4 Summary

This chapter has covered the fundamentals of using ODS styles with tagsets. The interaction is really quite simple and direct. As an Excercise try using one of the various tagsets in this chapter to create a style mapping tagset. Try to exclude events that do not have a style or vice versa.

Chapter 10

Tagsets with Style

The fundamentals of tagsets are behind us, now it is time do something with them. In this chapter we will do examples that specifically leverage ODS styles to accomplish their tasks.

10.1 A Problem with Table Rules

The tables that ods generates are fairly basic. Most people probably never take a second look at their style. Sometimes there are special needs for the way a table is done. Most often that has to do with the rules, or lines, that outline the table. Previous to SAS 9.1 ODS styles provided four attributes to control these, frame, rules, bordercolor and borderwidth. If those four don't do what you want then things are going to be more difficult.

10.1.1 A demonstration

10.1.2 A Simple Solution

10.1.3 Table Rules with style

10.1.4 The Code

10.2 Everyone likes stripes

Another common request is to create tables with striped rows. This is another thing that is quite easy to do with styles. We need a style to contrast the data style already in use. And we need to switch between the data style and the new style we create. This is especially easy in SAS 9.1 because of do blocks.

10.2.1 Defining the Problem

10.2.2 The HTML solution

10.2.3 The Code

10.2.4 The LaTeX solution

10.2.5 The Code

10.2.6 Summary

10.3 slidebar columns

This is a great example of what can be done with a little imagination. This example takes advantage of cell widths as percentages, to create a bar chart affect within a table column.

10.3.1 The mechanics

10.3.2 The Code

10.3.3 Summary

Chapter 11

Tagsets with Streams

Streams add another level of versatility to tagsets. Streams allow for output to be saved away, and reused or just delayed. This is ideal for solving problems where the ODS event model does not match the markup you are trying to create. In this chapter we will show how to use streams to solve problems that would otherwise be impossible.

11.1 Overly long Tables

The tables created by ods markup are long and continuous. Unlike the other ODS destinations markup does not panel it's tables into more manageable pieces. We can fix that with tagsets, by using streams.

11.1.1 The Solution

11.1.2 The Code

11.1.3 summary

11.2 Special Cases

The last tagset seems complete, but what happens when it is used with the Tabulate or Report procedures?

11.3 One Better

11.3.1 The solution

11.4 A Tagset with Startpage

Several of the ODS destinations have an option called startpage. Startpage has at least two options, on and off. On is normal behavior. Off means that pagebreaks, titles and footnotes are suppressed. There are several other possible values for startpage, the only one that makes much sense for a tagset is 'now'. Now means that the footnotes, pagebreak and titles should happen next.

11.4.1 The Mechanics

11.4.2 Block and Unblock statments

11.4.3 The code

11.4.4 The Code, Part 2

11.4.5 Another Event

11.4.6 Refining the Control

11.4.7 The code, part 3

11.4.8 The Final Cut

11.5 summary

Streams are one of the most powerful features available in tagsets. They can be used to delay and repeat output or rearrange output to fit your needs. Streams do expose some difficulties presented by specific procedures but with careful consideration, thought and maybe a little frustration, those problems can be overcome.

Part IV

Advanced Examples

Chapter 12

ODS Output for Website Integration

Many web sites have a framework that web pages must fit into before they will be published on the web. Usually this means editing your html pages and pasting in some chunk of HTML from some files your website authors have provided. Usually there is one file to replace the top section of your HTML and another to replace the bottom. We can create a tagset that will create HTML that is ready for integration with your website. This chapter will show you how.

12.1 The Problem

Most websites want your webpage to be enclosed with some sort of image and navigation menu at the top and/or bottom. Most of this is usually enclosed in some sort of table to control the layout.

12.2 Alternate Behavior for existing options

The first step towards our goal is to eliminate all the events above and below the body tags. ODS already gives us options to do that. All we need is to detect when those options are set. Then detect if there are files that can be used in place of the head and foot sections of the document.

12.3 Reading an external file

12.4 The Solution

The following tagset will read the files given through the tagset alias or a macro variable called infiles. Alias or infiles should be two filenames separated by a space. If tagset alias is set to 'default' then the set_default_files event is used to set the default file names. By using a separate event, this tagset can be used as a parent tagset, where only the set_default_files event is redefined. This also allows normal no_top behavior if no files are specified.

12.4.1 Initialization Timing

12.5 Summary

The output using these particular template files is shown in figure **??** on page **??**. Working with the authors of your website should enable you to create some very nice looking output. All of which is immediately ready for integration into your company's website.

Chapter 13

Datastep Conversions

You may wonder, why would someone want to convert a datastep program to use tagsets? There are lots of reasons. Mostly it's for code reuse and flexibility. Datastep programs allow lots of flexibility but they are written for one purpose. If the same functionality is desired for a different set of data an entirely new datastep will have to be written. If the rendering is left to a tagset, then anyone can use that tagset with any form of data, and any procedure. Maintenance of the code is also simplified since both the tagset and the resulting sas job wil be much simpler than the original datastep. Add the various powers of ODS to this new found flexibility and we have plenty of motivation for conversion. This chapter will show the process and rewards of converting datastep programs to tagsets.

13.1 Special Bylines

One of the things that can be done in datastep is the counting of observations for display in the byline. The datastep just counts them first then another datastep prints everything out. Tagsets can do that too. This first example has a special style, as well as a special byline. The byline text may change based upon the current by value. Each byline also displays the number of observations in the table below it.

13.1.1 The DataStep Code

13.1.2 Breaking it down

13.1.3 The Style

13.1.4 Counting Observations

13.1.5 various problems

13.1.6 Modifying the Byline

13.1.7 A more flexible solution

13.2 Slidebars for HTML, PDF, and PS

This example used datastep to create a table within each row of the data table. The result was a fancier version of the slidebar example on page **??**. The motivation for converting this datastep was to get PDF and postscript output. That is possible because of the LaTeX tagset. By creating both an HTML and LaTeX tagset all of these output types are possible.

13.2.1 breaking it down

13.2.2 The Style

13.2.3 The HTML Tagset

13.2.4 Dealing with the Report Procedure

13.2.5 The LaTeX Tagset

13.3 Summary

This chapter has shown some of the advantages of converting you datastep code to tagsets. Tagsets do not pretend to replace datastep, but if responsiblities are divided between datastep and tagsets then the solution is more open for re-use in later solutions. More power and flexibility is also provided by the tight integration the output now has with ODS.

Chapter 14

extended examples

This chapter will explore some more complex examples. The complexity comes mostly from the combination of the many techniques we have learned so far. From a juggler's point of view, we just have more balls in the air. But a ball is still just a ball. Any given feature of these new tagsets is still just as simple as it ever was. This is one of the wonderful things about tagsets. It is easy to reuse and combine ideas from different tagsets.

14.1 Repeating Headers, and Mirrored Row headers

When tabular output is extremely wide or long it is often hard to tell what the data is because the haders have scrolled out of view. This tagset solves that problem by putting row headers on both sides of the table. It also repeats the headers as in our previous example on page **??**

14.1.1 The Single Stream Solution

14.1.2 The Multiple Stream Solution

14.1.3 The Multiple List Solution

14.2 Automatic Panelling

In this example we will combine our startpage tagset with some a dictionary and some new events to create a tagset that automatically panels output.

14.2.1 An Extension of Start Page

14.2.2 The solution

14.2.3 Summary

Chapter 15

A feature Rich Tagset

In this book we have created many tagsets that can each do something special. The tagsets have been carefully written to enable those features to be transparent to anyone who uses them. It is possible that all of these features be combined into one tagset. So that is what we are going to do.

15.1 Which features?

two - sided, start page, sliders, table head style, stripes, nobs, byline, web site.

15.2 Lining up the inheritance

We can use some of our existing tagsets just by changing their inheritance. If we pick them carefully we won't have to change as many events.

15.3 Copy and Paste

It might just be easier to create one big tagset that does it all...

15.4 Macro Variables and Tagset Alias

Turning all of these things on and off presents some special problems. Each of them want macro variables, tagset alias or both.

15.5 Summary

Pretty Cool.

Part V

Usage Notes and Caveat’s

Chapter 16

Special Cases, The Report and Tabulate Procedures

We have seen, in our examples, that there are some problems when it comes to the Report and Tabulate procedures. These problems are surmountable but they do cause some pain. This chapter is going to explain the specifics of these problems and how to work around them.

16.1 A Report Procedure problem

Yet another reason to convert is that tagsets can be interchanged to create new and diffrent outputs. Usually datastep is used to create HTML, CSV,

16.1.1 deferred data

16.2 The Tabulate and Report problem

More reasons to convert are the availability of style changes, and the plethora of ODS options that can be used to customize your output.

16.2.1 The Table Head section

16.2.2 The Table column specifications

Chapter 17

Using LaTeX

Besides HTML and XML, LaTeX output is one of the most useful and versatile output destinations available. LaTeX is commonly used in publishing and is capable of creating several viewable formats, including PDF and Postscript. LaTeX supports color and various image formats. Sadly, this destination is largely overlooked. This chapter will discuss how to use it to your advantage.

17.1 The LaTeX statement

Latex is one of the ’special’ destination names that is shorthand for ’tagsets.latex’. But there are other latex tagsets. Color_latex differs only in the way it invokes the usepackage statement to include the ’stylesheet’ generated by latex. Simple latex is the most simplistic form of latex which lends itself to embedding in other LaTeX documents.

17.1.1 Color

17.2 Compiling the LaTeX Output

There is no browser for LaTeX. LaTeX must be compiled into the document type desired. Different commands do different things.

17.2.1 The latex Command

17.2.2 The dvi2ps Command

17.2.3 The pdflatex Command

17.3 Integrating LaTeX output into documents

Aside from creating pdf or postscript reports it is sometimes desirable to create output that will be used in a larger LaTeX document. A paper or book for example.

17.3.1 The easy way

17.3.2 The simple way

17.3.3 Using NewFile to advantage

17.4 LaTeX in the different versions of SAS

17.4.1 SAS 8.2

While ODS Markup was experimental in SAS 8.2 it was still a fairly stable product. The biggest exception to that was the LaTeX output. Embedded stylesheets was not an available feature in ODS Markup at in that release. An external stylesheet had to be used.

17.4.2 SAS 9.0

17.4.3 SAS 9.1 and beyond.

17.5 Summary

LaTeX is one of the most useful outputs that ODS creates. It can be easily be used within Documents of any size from reports, to Papers to books. It can be used to create many output types including PDF, Postscript, and DVI.

Chapter 18

Using The HTML Tagsets

There are several HTML tagsets to choose from. Each generates a slightly different type of HTML. Each have their advantages. This chapter will explain the different HTML tagsets and the features they support.

18.1 The html statement

html is one of the ’special’ destination names that is shorthand for ’tagsets.html4’. But there are other special names as well. PHTML, HTMLCSS, MSOffice2K and Chtml are the other shortcut names for HTML tagsets.

18.1.1 HTML4

18.1.2 PHTML

18.1.3 HTMLCSS

18.1.4 MSOffice2K

18.2 CHTML

There is no browser for LaTeX. LaTeX must be compiled into the document type desired. Different commands do different things.

18.3 Stylesheets

The best way to use them.

18.4 Javascript code

Where to put it?

18.5 Scrolling Tables

Post SAS 9.1 the html tagsets received a new feature, Table scrolling. This feature is part of the tagset download on the ODS markup Resources web page.

18.6 HTML and Excel

HTML is one of the better ways to import SAS ouput into excel. But some tagsets work better than others....

Chapter 19

Using Markup output with Spreadsheets.

There are various ways to get SAS output into a spreadsheet. Each have their advantages and disadvantages. This chapter will go over the various tagsets that good for importing into spreadsheet programs.

19.1 Using CSV

Comma separated values are one of the simplest formats that are is understood by spreadsheet programs. The disadvantage is that there is no formatting, fonts, or colors. Still CSV files work very well.

19.2 SYLK

Symbolic Link format is another simple format that is well recognized by spreadsheet programs. Sylk is slightly better than CSV because it does contain field formats and data types.

19.3 HTML and Excel

HTML is one of the better ways to import SAS ouput into excel. But some tagsets work better than others....

19.3.1 Compact HTML

19.3.2 PHTML - Plain HTML

19.3.3 HTML for Microsoft Office 2000

19.4 DDE

19.5 Spreadsheet XML

The ExcelXP tagset generates Spreadsheet XML. It is probably the best format for importing SAS ouput into Excel. It does have some limitiations though. It can only have one table per worksheet. And it does not support graphics. Both of those things work perfectly fine when importing HTML.