ODS Markup:
Tagsets by Example

Eric Gebhart
SAS Institute

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 output can be modified and adapted in ways that could only be imagined until now. 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.

Contents

I Using ODS Markup and Tagsets **1**

1 Introduction **3**

2 The basics **5**

2.1 It's all in the Name 5

2.1.1 The tagset directory 6

2.2 Using a tagset 7

2.3 Summary 9

3 Using The CSV Destinations. **11**

3.1 Summary 16

4 Using The HTML Tagsets **19**

4.1 The Different HTML's 19

4.1.1 HTML4 And XHTML 20

4.1.2 HTMLCSS 20

4.1.3 PHTML 20

4.1.4 XHTML 21

4.1.5 MSOffice2K 21

4.1.6 HTML4 21

4.1.7 PHTML 21

4.1.8 HTMLCSS 21

4.1.9 XHTML 21

4.1.10 MSOffice2K 21

4.2 HTML and Excel 22

4.3 CHTML 22

4.3.1 CHTML_imode 22

4.4 Stylesheets 22

4.5 Javascript code 22

4.6 Accessibility 23

4.7 Summary 23

5 Using LaTeX **25**
5.1 The LaTeX statement . . . 25
5.1.1 Color Support . . . 25
5.2 Compiling the LaTeX Output . . . 26
5.2.1 The latex Command . . . 26
5.2.2 The dvips Command . . . 26
5.2.3 The pdflatex Command . . . 27
5.3 Integrating LaTeX output into documents . . . 27
5.3.1 The easy way . . . 27
5.3.2 Using NewFile to advantage . . . 27
5.3.3 The simple way . . . 28
5.4 Image Formats and Graph . . . 28
5.5 LaTeX in the different versions of SAS . . . 29
5.5.1 SAS 8.2 . . . 29
5.5.2 SAS 9.0 . . . 29
5.5.3 SAS 9.1 and beyond. . . . 29
5.6 Summary . . . 30

II Beginning Tagsets **31**

III Intermediate Examples **33**

IV Usage Notes and Caveat's **35**

V Appendices **37**

Quick Reference Guide **39**
.1 Useful tagsets . . . 39
.2 Tagset attributes . . . 41
.3 Event attributes . . . 44
.4 Event Statements . . . 44
.5 If Statements . . . 47

Variables **49**
.6 Event Variables . . . 49
.6.1 508 Accessibility . . . 49
.6.2 Data . . . 49
.6.3 Data Formatting . . . 50
.6.4 Event MetaData . . . 51
.6.5 Graph . . . 51
.6.6 Measured . . . 51
.6.7 Miscellaneous . . . 52
.6.8 ODS Statement . . . 53

.6.9 Table 55
.6.10 Title and Note Formatting 55
.6.11 URL 56
.6.12 XML Libname Engine 56
.7 Style Variables 56
.7.1 Borders 56
.7.2 Font 57
.7.3 Background 58
.7.4 Frame 58
.7.5 Miscellaneous 59
.7.6 Graph 60

List of Tables

2.1 Shortcut Destination names 9

Part I

Using ODS Markup and Tagsets

Chapter 1

Introduction

ODS markup and tagsets are the best way to create markup output from SAS. This book is divided into sections that will enable you to get the most out of ODS Markup and Tagsets. The first section shows how ODS Markup and Tagsets are used. Using an ODS destination that is defined by a tagset is no harder than using any other destination. The most popular ODS Markup destinations are covered in this first section.

The next sections dig deeper into how Tagsets work and what you can do to modify them or create your own. This may seem daunting at first, but frequently it is quite easy to get Tagsets to do exactly what you want, saving lots of time and effort further down a project's path.

Chapter 2

The basics

This chapter will cover the basics of using ODS Markup family of destinations. There are a lot of ODS destinations in this family, and they are ready to use. By the end of this chapter you will have some idea of the destinations available, and how to use them.

ODS Markup is just another ODS destination that works very much like every other ODS destination. It was originally derived from the original ODS HTML destination, so if you are familiar with ODS HTML, then you are already very familiar with ODS Markup.

The way ODS Markup differs is that we will almost never use ODS Markup as a destination name. ODS Markup should be thought of as a family of destinations. These destinations are defined using a type of template called Tagsets. Each Tagset definition is a new ODS destination.

The good news is that there are already a lot of Tagsets defined, so all we need to do is use them. Since Tagsets define destinations within the ODS Markup family the only thing we really need to know is their names. Once we know the tagset's name we know the destination name, and we can easily construct an ODS statement that will use that ODS destination.

2.1 It's all in the Name

A tagset is a SAS program that uses Proc Template. There are a lot of advantages to this. It means that SAS can update a tagset, make a fix, and even email that tagset to anyone that wants it. Updates to the Tagsets that shipped with SAS are available for download on the Support.sas.com website. Frequently there are also additional tagsets that were not shipped with SAS. Another benefit, is that if you want to learn to program Tagsets, you can create your own, or modify the Tagsets provided by SAS.

For now, the most important thing to know, is what are the Tagsets available, and what are they named. All we really need is their names. But where do we look? When a Tagset is run in SAS, Proc Template compiles the Tagset and stores it When a Tagset is run in SAS, Proc Template the Tagset 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. All Tagsets that ship with SAS are in SASHELP.

Thankfully, Proc Template handles all of this for us, and will look for Tagsets in all the right places. All we need is 3 lines of SAS code, and we will have all the names for all the Tagsets.

2.1.1 The tagset directory

Templates are kept in directories within the itemstore. ODS will automatically look for tagsets in the tagsets directory. The following example shows how to get a list of tagsets, and the output it generates.

Listing 2.1: Simple ODS Markup Statement

```
proc template;
    list tagsets;

                                The SAS System
1
                                                    13:00 Monday, August 7, 2006

                       Listing of: SASHELP.TMPLMST
                       Path Filter is: Tagsets
                       Sort by: PATH/ASCENDING

                       Obs    Path                              Type
                        1     Tagsets                           Dir
                        2     Tagsets.Cascading_stylesheet      Tagset
                        3     Tagsets.Chtml                     Tagset
                        4     Tagsets.Colorlatex                Tagset
                        5     Tagsets.Config_debug              Tagset
                        6     Tagsets.Csv                       Tagset
                        7     Tagsets.Csvall                    Tagset
                        8     Tagsets.Csvbyline                 Tagset
                        9     Tagsets.Default                   Tagset
                       10     Tagsets.Docbook                   Tagset
                       11     Tagsets.Event_map                 Tagset
                       12     Tagsets.ExcelXP                   Tagset
                       13     Tagsets.Graph_rtf                 Tagset
                       14     Tagsets.Html4                     Tagset
                       15     Tagsets.Htmlcss                   Tagset
                       16     Tagsets.Htmlpanel                 Tagset
                       17     Tagsets.Imode                     Tagset
                       18     Tagsets.Latex                     Tagset
                       19     Tagsets.MSOffice2k                Tagset
                       20     Tagsets.Mlatex                    Tagset
                       21     Tagsets.Mvshtml                   Tagset
                       22     Tagsets.Namedhtml                 Tagset
                       23     Tagsets.Odsapp                    Tagset
                       24     Tagsets.Odsgraph                  Tagset
                       25     Tagsets.Odsstyle                  Tagset
                       26     Tagsets.Odsxrpcs                  Tagset
                       27     Tagsets.OpenOffice_rtf            Tagset
                       28     Tagsets.Phtml                     Tagset
                       29     Tagsets.Pyx                       Tagset
                       30     Tagsets.Rtf                       Tagset
                       31     Tagsets.Rtf_sample                Tagset
                       32     Tagsets.SASReport11               Tagset
```

```
                33     Tagsets.SASReport12              Tagset
                34     Tagsets.SASReport13              Tagset
                35     Tagsets.SASReport14              Tagset
                36     Tagsets.SASReport15              Tagset
                37     Tagsets.SASReport_html           Tagset
                38     Tagsets.SASReport_html1          Tagset
                39     Tagsets.Sasreport_html11         Tagset
                40     Tagsets.Short_map                Tagset
                41     Tagsets.Simplelatex              Tagset
                42     Tagsets.Style_display            Tagset
                43     Tagsets.Style_popup              Tagset
                44     Tagsets.Supermap                 Tagset
                45     Tagsets.Tablesonlylatex          Tagset
                46     Tagsets.Text_map                 Tagset
                47     Tagsets.Tpl_style_list           Tagset
                48     Tagsets.Tpl_style_map            Tagset
                49     Tagsets.Troff                    Tagset
                50     Tagsets.Wml                      Tagset
                51     Tagsets.Wmlolist                 Tagset
                52     Tagsets.Xbrl                     Tagset
                53     Tagsets.Xhtml                    Tagset
    run;

NOTE: PROCEDURE TEMPLATE used (Total process time):
      real time           1:40.15
      cpu time            0.28 seconds
```

2.2 Using a tagset

That is a lot of Tagsets! For now, lets concentrate on the obvious ones. To use a tagset, all that is needed is the 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 2.2 on page 7.

Listing 2.2: Simple ODS Markup Statement

```
ods markup file='test.xml';

....

ods markup close;
```

The result of those statements will be the creation of the 'test.xml' output file. The output of that file will have a format as defined by the default tagset, which is named, tagsets.default.

But there are other ways to specify an ODS statement that accomplish the same thing. Figure 2.3 on page 7. shows ods statements which are equivalent with one exception.

Listing 2.3: ODS Markup Statement Permutations

```
\index{ods markup statement!permutations}

    ods markup file='test.xml';

    ods markup tagset=default file='test2.xml';

    ods tagsets.default file='test3.xml';
```

The exception is the third statement. That statement is different because the destination name is no longer markup. The destination name is tagsets.default. Because it has a unique name it can run simultaneously with ods markup. Just like ODS RTF can run simultaneously with ODS HTML. The first two statements cannot be used simultaneously because they have the same destination name. The second ODS Markup statement would automatically close the first one. This happens because ODS destination names must be unique. There is a way around that, using ODS' ID syntax, but that belongs in another book. Beside's that syntax is mostly uneccessary if we use the tagset name as the destination name, and the resulting code is more readable and concise.

As of SAS 9.1, ODS HTML is also a tagset. Which means ODS HTML is really ODS Markup. Consider the statements in figure 2.4 on page 8.

Listing 2.4: ODS Markup Statement Permutations

```
ods html file='test1.html';

ods html4 file='test2.html';

ods tagsets.html4 file='test3.html';

ods markup tagset=html4 file='test4.html'
```

Each one of those statements is roughly equivalent. All of them use the html4 tagset. So the markup each of them creates will be the same. But each ODS statement creates a unique output destination. They can all run simultaneously without conflict. They can all have their own select or exclude statements and be opened or closed at different times.

There is one thing that makes html special. Normally a tagset cannot be referenced by it's simple name unless the tagset option is used. But there are a few special tagsets that can be referenced as if they are a simple ODS destination. Table 2.1 on page 9 shows the list of shortcut names and their descriptions.

While these are special, the custom tagsets that can be written are no less special. These custom tagsets cannot be referenced by their simple names but they will be output destinations just as these are. Because Markup is a single destination, the prefered way to use tagsets is to use their two part name. There is a trend away from using these shortcut names, so most of the newer tagsets like ExcelXP and RTF can only be referenced by their full name. See Figure 2.5 on page 8 shows the list

Listing 2.5: Multiple Active Markup Destinations

```

ods tagsets.ExcelXP file='test.xml';
ods tagsets.RTF file='test.rtf';

proc print data=sashelp.class;
run;
```

Table 2.1: Shortcut Destination names

HTML	Alias for HTML4
HTML4	4.0 compliant HTML with concessions for browser compatibility.
PHTML	Plain HTML. Simple stylesheet, simple HTML.
HTMLCSS	4.0 compliant HTML with fewer concessions.
CSV	Comma separated values. Tables only.
CSVAll	CSV with titles, bylines, notes, etc.
CSVByline	CSV with bylines only.
WML	Wireless Markup Language. - No longer in vogue.
CHTML	Compact HTML. Very simple HTML with no styles.
LaTeX	LaTeX output with style and color support.
Troff	troff output which is very simple.
MSOffice2K	HTML for importing into Word and Excel.
Imode	Compact HTML that has no table tags. Used extensively in Japan.
DocBook	XML format for documents.
SasReport	XML format used internally by other SAS products.

```
ods tagsets.ExcelXP close;
ods tagsets.RTF close;
```

2.3 Summary

Using ODS Markup does not have to be complicated or difficult, all we really need is the name of the Tagset, and we can use them just like any other destination. Finding their names is very simple Proc Template code, and adding an ods statement that uses them is just as easy. It won’t hurt anything to try out a destination just to see what it will create. Go ahead, play, try them out. The next chapters will go into more detail for many of the more popular and useful Tagsets that are available.

The most confusing thing about the ods markup statement is that it is almost never used by it’s name. Aside from that it is almost the same as any other ods destination, especially the ODS HTML destination in previous releases of SAS.

Chapter 3

Using The CSV Destinations.

The CSV Tagsets are some of the most useful tagsets around, the format has been used for decades by various applications and everyone knows what to expect from a CSV file. This chapter will explain how to use the 3 different CSV tagsets provided by SAS.

Comma separated values are one of the simplest formats that are is understood by many different applications. The disadvantage is that there is no formatting, fonts, or colors. Still CSV files work very well for many applications. A CSV file can be connected as live data to an Excel Template. This can be a convenient way to publish Excel spreadsheets to many people.

There are three different CSV Tagsets shipped with SAS. The simplest is the CSV Tagset, next comes CSVByline, and last is CSVAll. The CSV Tagset only creates output from tabular data, there are no titles, footnotes, bylines, Notes, there is nothing but tables. It didn't take long to find out that there were people who wanted titles and footnotes and everything else, that's when CSVAll came into existence. It also became apparent that just having tables and bylines was a nice thing to have, so CSVBylines was born. All of these tagsets use the same base Tagset to create the CSV data, so there is no other difference in behavior besides the verbosity of the output.

Both CSV and CSVAll have aliases so that they can be referenced by a simple name, rather than the full tagset name. CSVByline has no alias, so it must be used with it's full name. Here are examples of each of these ODS statements.

```
ODS CSV file='test.csv';
ODS CSVAll file='testall.csv';
ODS tagsets.CSVByline file='testby.csv';
```

Using these statements around a simple Proc Print will show how these destinations vary.

```
ODS CSV file='test.csv';
ODS CSVAll file='testall.csv';
ODS tagsets.CSVByline file='testby.csv';

options obs=6;

proc sort data=sashelp.class out=sorted;
     by sex;

Proc print data=sorted;
```

```
        by sex;
run;

ODS _all_ close;
```

None of the output from this example is very complicated, but each has it's differences. The CSV Tagset ignores the title and the bylines. But it does create the two tables as a result of the by.

```
"Obs","Name","Age","Height","Weight"
"1","Alice",13,56.5,84.0
"2","Barbara",13,65.3,98.0
"3","Carol",14,62.8,102.5

"Obs","Name","Age","Height","Weight"
"4","Alfred",14,69.0,112.5
"5","Henry",14,63.5,102.5
"6","James",12,57.3,83.0
```

The CSVByline Tagset adds the bylines, but nothing else.

```
"Sex=F"
"Obs","Name","Age","Height","Weight"
"1","Alice",13,56.5,84.0
"2","Barbara",13,65.3,98.0
"3","Carol",14,62.8,102.5

"Sex=M"
"Obs","Name","Age","Height","Weight"
"4","Alfred",14,69.0,112.5
"5","Henry",14,63.5,102.5
"6","James",12,57.3,83.0
```

Finally, the CSVAll tagset adds in the title. If there were footnotes, or procedure titles, or any of the four flavors of notes; Note, Warning, Error, or Fatal, those would be included as well.

```
The SAS System

"Sex=F"
"Obs","Name","Age","Height","Weight"
"1","Alice",13,56.5,84.0
"2","Barbara",13,65.3,98.0
"3","Carol",14,62.8,102.5

"Sex=M"
"Obs","Name","Age","Height","Weight"
"4","Alfred",14,69.0,112.5
"5","Henry",14,63.5,102.5
"6","James",12,57.3,83.0
```

In the beginning it seemed that this was everything that any CSV tagset would ever need to do. But even the simplest of outputs is not always that simple. Tagsets needed to be able to adapt to what was needed of them. The best way to do that is to have the ability to give the tagset some variable to help it make decisions on how to behave. Up until SAS 9.1.3 the only way to do that was with macro variables. Macro variables still work, but now there is a better way. With SAS 9.1.3 the ODS Markup statement was given a new option, the options(...) option. Options are a nearly arbitrary set of key value pairs that the tagset will recieve. It is completely up to the Tagset to process those values and use them.

Surprisingly the CSV Tagset has quite a number of options. In fact everything that the CSVAll and CSVByline tagsets do can be done by the base CSV Tagset. In SAS 9.1.3 the only thing CSVALL and CSVByline Tagsets do is set up some new defaults for the CSV options available.

With the proliferation of Tagset options it didn't take long to figure out we needed another option, Help! All Tagsets that have options have an option called doc. The Doc option is all you need to find out just what a Tagset can do. Asking for help may be the first thing you want to do when using a new tagset. If there are options, Help will tell you about them. Help isn't in all of the Tagsets, but the number is growing and it never hurts to ask. Here is how we ask for help.

```
ODS CSV file='test.csv' options(doc='help');
```

The output from help will go to the log and for CSV, it looks like this.

```
==============================================================================
The CSV Tagset Help Text.

This Tagset/Destination creates output in comma separated value format.

Numbers, Currency and percentages are correctly detected and show as numeric values.
Dollar signs, commas and percentages are stripped from numeric values by default.

==============================================================================

These are the options supported by this tagset.

Sample usage:

ods csv options(doc='Quick');

ods csv options(currency_as_number='yes' percentage_as_number='yes' delimiter=';');

Doc:  No default value.
     Help: Displays introductory text and options.
     Quick: Displays available options.

Delimiter:   Default Value ','
     Sets the delimiter for the values.  Comma is the default.  Semi-colon is
     a popular setting for european sites.

currency_as_number:   Default Value 'No'
     If 'Yes' currency values will not be quoted.
     The currency values are stripped of punctuation and currency symbols
     so they can be used as a number.

percentage\_as\_number:   Default Value 'No'
     If 'Yes' percentage values will not be quoted.
     The percentages are stripped of punctuation and the percent sign
     so they can be used as a number.
```

```
Currency\_symbol:   Default Value '\$'
     Used for detection of currency formats and for
     removing those symbols so excel will like them.
     Will be deprecated in a future release when it is
     no longer needed.

Decimal\_separator:   Default Value '.'
     The character used for the decimal point.
     Will be deprecated in a future release when it is no longer needed.

Prepend\_Equals:   Default Value 'no'
     Put an equal sign in front of quoted number values.
     This only works in conjunction with quote_by_type.

quote\_by\_type:   Default Value 'no'
     Put values based on the type, not based on what the regex match for number.

Table\_Headers:   Default Value 'yes'
     If no, skip the header section of all tables.

Thousands\_separator:   Default Value ','
     The character used for indicating thousands in numeric values.
     Used for removing those symbols from numerics so excel will like them.
     Will be deprecated in a future release when it is no longer needed.

Quoted\_columns:   Default Value ''
     A list of column numbers that indicate which values should be quoted
     ie. Quoted\_columns="123"

Empty\_Missing:   Default Value 'no'
     If yes, missing values will not show in any way other than positionally.

Bylines:   Default Value: No
     If yes bylines will be printed

Titles:   Default Value: No
     If yes titles and footnotes will be printed

Notes:   Default Value: No
     If yes Note, Warning, Error, and Fatal notes will be printed

Proc\_Titles:   Default Value: No
     If yes titles generated by the procedures will be printed
```

The help you get will depend upon which version of the tagset you have. There are the expected options that reflect the behaviors we have seen from the CSVAll and CSVByline Tagsets, but there is much more. One of the most used options is Delimiter, this makes it easy to change the field delimiter to a ;, a | or even tabs. Another thing that became obvious was that there needed to be a way to detect numbers in any one of the various European formats. That's where decimal_separator, thousands_separator and currency_symbol come in. Not everyone uses a '.' for decimals, or ',' for thousands.

There are a few less obvious options in there. Table_headers turns the table_headers on and off. Empty_missing makes missing values go completely away, with just it's field delimiters showing where it would have been.

Currency_as_Number, and Percentage_as_Number, are there because not everyone wants those values as strings, So if you want them as numbers, these options will do that for you, they will also strip out any symbols that would prevent Excel from seeing those values as numbers.

Another option that is just for excel, is prepend_equals, this option will put an '=' in front of quoted numbers so that Excel will read the value correctly, even if it has leading zeros. So if you have leading zero's this is the option you want.

The last two have to do with quoting. Currently, the CSV tagset tries it's best to figure out what is a string and what is a number. It doesn't always do exactly what we want. Part of the problem is that some procedures don't tell the Tagset what is what. Proc Print is one of those procedures that does say which value is a string and which is a number. The quote_by_type option tells the tagset to stop guessing and do what Proc Print or any other proc says. The exception to this, until SAS 9.2, is Proc Report, and Proc Tabulate. Those are the two Procs that don't give the tagset any clues.

To further extend the ability to control quoting, There is a Quoted_columns option that will allow for columns to be specified by number. So if all else fails and you still can't get your values quoted where you want, this option should do the trick.

By default the csv tagset only displays tables. Turning on bylines so that it looks like the output from the CSVByline Tagset is just a matter of setting the bylines option to yes. Changing the Delimiter to a ';' is just as easy. Of course we could have used the CSVByline Tagset with just the delimiter option to do the same thing.

```
ODS CSV file='test.csv' options(delimiter=';' bylines='yes');
ODS tagsets.CSVByline file='test.csv' options(delimiter=';');

options obs=6;

proc sort data=sashelp.class out=sorted;
     by sex;

Proc print data=sorted;
     by sex;
run;

ODS _all_ close;
```

The output from both of these ODS statements looks identical.

```
"Sex=F"
"Obs";"Name";"Age";"Height";"Weight"
"1";"Alice";13;56.5;84.0
"2";"Barbara";13;65.3;98.0
"3";"Carol";14;62.8;102.5

"Sex=M"
"Obs";"Name";"Age";"Height";"Weight"
"4";"Alfred";14;69.0;112.5
"5";"Henry";14;63.5;102.5
"6";"James";12;57.3;83.0
```

Knowing how the CSVByline and CSVAll Tagsets work can be quite handy, Especially if you find that you always want the same basic set of options. It is much easier to create your own Tagset than it is to always type in the same options over and over. Here is the entire code for the CSVByline Tagset. Before you look, realize that most of this is copied and pasted from the CSV Tagset.

```
define tagset tagsets.csvbyline;
    parent=tagsets.csv;

     define event initialize;
         set \$options['BYLINES'] 'yes';
         trigger set_options;
         trigger documentation;
         trigger compile_regexp;
     end;
 end;
```

The only addition to the copied and pasted code is this line.

```
set \$options['BYLINES'] 'yes';
```

We can take advantage of this knowledge and create our own CSV Tagset that has it's own set of default option values. If we want the same behavior from the new Tagset as our previous example a new name is probably in order. After all it's no longer comma separted values, it's semi-colon separated values. Our new Tagset would look like this. Be aware that option names are always capitalized inside the tagset.

```
proc template;
    define tagset tagsets.ssvbyline;
        parent=tagsets.csv;

         define event initialize;
             set \$options['BYLINES'] 'yes';
             set \$options['DELIMITER'] ';';
             trigger set_options;
             trigger documentation;
             trigger compile_regexp;
         end;
     end;
run;
```

All we have to do is run that Proc Template, and our new tagset is ready to use. Congratulations! You've just written your first Tagset! If someone else wants output that has bylines and semi-colons for field separators, just tell them to do this!

```
ODS tagsets.ssvbyline file='test.csv' options(doc='help');
```

The output looks exactly like the output above. It is, after all, generated the same way.

3.1 Summary

The CSV destinations are very easy to use and although the output is very simple, there are many options that can be used to control how that output is created. ODS Markup options provide a very versatile way to change the behavior of any Tagset destination. Finding out what those options are, and how to use them is as easy as asking for help. Add options(doc='help') to any ODS Markup/tagsets statement and find out just what that Tagset can do for you. Creating a new Tagset that sets the options just the way you want is almost as easy is copy and paste. That's what tagsets are all about, creating output just the way you want it, in a reusable and easy way.

Chapter 4

Using The HTML Tagsets

HTML is certainly one the most popular output typs available. Not only is easily used to report information on the web, but it is also a commonly understood import format for many applications. ODS Markup has several HTML tagsets to choose from when it comes to creating HTML output. Each of these Tagsets has some advantage in one way or another depending upon how it will be used. This chapter will explain the different HTML tagsets and the features they support.

4.1 The Different HTML's

ODS HTML is one of the aliased destination names that really means use the tagsets.html4 Tagset. Someday it may well mean use the Tagsets.html5 Tagset. But that is in the future.

There are several other aliases, for the various HTML tagsets. PHTML, HTMLCSS, CHTML and CHTML_imode are the other shortcut names for HTML tagsets. These aliases map directly to their tagset names, the aliases just keep us from having to put 'tagsets.' in front of them.

Only one example is needed in order to see the major differences between the different HTML tagsets available. There are differences in style, the way titles and footnotes are processed, and in the case of the MSOffice2k tagset, there is a difference in how the tables are rendered. All that is needed to see this differences is a couple of title statements and some simple Proc Tabulate output.

This example has three titles, the default title, and two more titles that use the 3 part title syntax. It is the use of style and 3 part titles that will show major differences between these various HTML destinations.

```
ods html file="html.html";
ods xhtml file="xhtml.html";
ods htmlcss file="htmlcss.html";
ods phtml file="phtml.html";
ods tagsets.MSOffice2K file="msoffice2k.html";
ods chtml file="chtml.html";
ods imode file="imode.html";


  title3 height=15pt color=orange
         justify=right '15 pt first right'
         justify=left 'first left';
```

```
    title4 height=10pt color=red 'no justify red'
           justify=right 'Second right'
           justify=left 'Second left';

  proc sort data=sashelp.class out=work.class;
      by age  sex;
  run;

  options obs=2;

  PROC TABULATE DATA=class;
     VAR Height Weight;
     CLASS Sex Age;
     TABLE Age*Mean ALL*Sex*Mean,
           Weight;
     by age;

  RUN;

  ods _all_ close;
```

4.1.1 HTML4 And XHTML

HTML4 and HTML are really the same thing and XHTML is nearly the same but not quite. The only thing the XHTML tagset does is add XML compliant tags to the HTML. If you don't know what that is, don't worry about it. For all practical purposes they are nearly identical.

The HTML destination does not have to be HTML4, it could be set to be HTML3. There is a registry setting that will allow HTML to be the legacy version HTML. HTML3 is the name of what was the HTML destination in SAS 8.2. HTML3 does not use tagsets or ODS Markup at all. It is done the old fashioned way with C-code. HTML3 has remained virtually unchanged since SAS 8.2.

Let's look at the HTML4 output first since HTML4 is the tagset that does everything. There aren't any surprises here. It looks just the way we would expect. The titles are colored, and justified on the page just the way they should be.

IMAGE MISSING....

Use this output as a reference as you look at the output for the rest of these HTML destinations. It will give you a good idea of how they all behave.

4.1.2 HTMLCSS

HTMLCSS is a basic stylesheet based HTML tagset. There is very little difference between the HTML4 tagset and HTMLCSS. Both the HTML4 and MSOffice2k tagsets inherit most of their events from the HTMLCSS tagset, and ultimately the PHTML tagset.

4.1.3 PHTML

The PHTML tagset generates a very plain HTML. The huge stylesheet is replaced by a much smaller stylesheet. If you dislike all the styles and the verbosity of the html4 and htmlcss tagsets then this tagset is a good place to start.

4.1.4 XHTML

The XHTML tagsets is one of the shortest tagsets around. It inherits from the HTML4 tagset. All it adds are the stricter XML style tags.

4.1.5 MSOffice2K

The MSOffice2k tagset is a good tagset to use if you want to import the resulting HTML into Microsoft Office. This tagset adds Microsoft specific XML to the HTML so that Microsoft office can size images better. It also creates different HTML for titles and footnotes so that they import more smoothly into Excel. This tagset also works with proc tabulate to create a better array of table cells, so that Excel will keep all of the rows straight.

4.1.6 HTML4

The HTML tagsets that use styles always use a stylesheet, if no stylesheet file is specified the stylesheet is still written to the top of the body file. Using an external stylesheet allows more flexibility and it can allow multiple reports to use the same stylesheet file.

```
ODS html file='test.html';

proc print data=sashelp.class; run;

ODS html close;
```

4.1.7 PHTML

The PHTML tagset generates a very plain HTML. The huge stylesheet is replaced by a much smaller stylesheet. If you dislike all the styles and the verbosity of the html4 and htmlcss tagsets then this tagset is a good place to start.

4.1.8 HTMLCSS

HTMLCSS is a basic stylesheet based HTML tagset. There is very little difference between the HTML4 tagset and HTMLCSS. Both the HTML4 and MSOffice2k tagsets inherit most of their events from the HTMLCSS tagset, and ultimately the PHTML tagset.

4.1.9 XHTML

The XHTML tagsets is one of the shortest tagsets around. It inherits from the HTML4 tagset. All it adds are the stricter XML style tags.

4.1.10 MSOffice2K

The MSOffice2k tagset is a good tagset to use if you want to import the resulting HTML into Microsoft Office. This tagset adds Microsoft specific XML to the HTML so that Microsoft office can size images better. It also creates different HTML for titles and footnotes so that they import more smoothly into Excel.

This tagset also works with proc tabulate to create a better array of table cells, so that Excel will keep all of the rows straight.

4.2 HTML and Excel

HTML is one of the better ways to import SAS ouput into excel. But some tagsets work better than others. HTML that uses H tags for titles, notes and bylines works better than HTML that uses tables for their formatting. That makes the HTML4 and HTMLCSS tagsets bad choices for importing into Excel. The Compact HTML, PHTML, and the MSOffice2k tagsets work the best.

4.3 CHTML

Compact HTML is a subset of HTML. Chtml has no style, it has a limited but very useful set of HTML tags. Chtml output is viewable on any browser. It is intended for use with phones and PDA's where download size is important.

4.3.1 CHTML_imode

Imode HTML is a subset of Compact HTML. Imode has no support for tables. It is primarily used for phones in Japan.

4.4 Stylesheets

If you choose to use stylesheets it can be convenient to have one stylesheet that everyone uses. It is easy to reference another stylesheet on the ODS Statement. This first ODS statement will create a stylesheet file called test.css.

```
ODS HTML file='test.html' stylesheet='test.css';
```

This next ODS statement does not create a stylesheet file at all. But it does put in a stylesheet link to the test.css stylesheet that was created earlier.

```
ODS HTML file='test.html' stylesheet=(url='test.css');
```

4.5 Javascript code

If you have a need for javascript in your HTML there are some good places to put that code. There is an event in the tagset called code_body that will write to the code file specified on the ods statement. A link to that file will be created in the body, contents, and pages files. It is also possible to trigger the code_body event when no code file is specified, that will allow support for both external and embedded javascript.

4.6 Accessibility

The HTML tagsets have accessibility features that are 508 compliant to level two. Level 3 compliance is possible with most procedures but will require updates to the table templates used by the procedures. Attributes in the table templates provide for summary, abreviation, acronym, long_description, caption, and alt. Setting these table and column attributes can improve your HTML's 508 compliance dramatically.

4.7 Summary

There are several HTML tagsets provided by SAS. Most likely one of them will be reasonably close to the style of HTML you wish to create.

Chapter 5

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 advantage.

5.1 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.

The best way to use the latex tagsets is with an external stylesheet. Using an external stylesheet is the only way to turn color on. The stylesheet includes usepackage statements for all the packages needed to render the latex. It also defines macros for the styles and formatting.

Another requirements of using latex is that a url without the .sty extension be specified for the stylesheet. This is the name that will go in the usepackage statement in the preamble of the latex document. A sample ods latex statement is shown here.

```
ods latex file="test.tex" stylesheet="test.sty"(url="test");
```

5.1.1 Color Support

Color LaTeX is really the same as the regular latex tagset. The only difference is that a flag is set so that colors will be used. The entire tagset looks like this. The change is the addition of the color option.

```
define tagset tagsets.colorlatex;
    parent=tagsets.latex;

    define event stylesheet_link;
        put CR '\usepackage[color]{';
        put URL;
        put '}' CR CR;
```

```
            end;

        end;
```

A better color latex tagset would work without a stylesheet url being specified on the ods statement. This is only possible in SAS 9.1, but is a perfect example of how useful data step functions can be.

```
proc template;
    define tagset tagsets.colorlatex;
        parent=tagsets.latex;

        define event stylesheet_link;
            put CR '\usepackage[color]{';
            put scan(url, 1, '.');
            put '}' CR CR;
        end;

    end;
run;

ods tagsets.colorlatex file="test.tex" stylesheet="test.sty";
proc print data=sashelp.class;run;
ods _all_ close;
```

5.2 Compiling the LaTeX Output

There is no browser for LaTeX. LaTeX must be compiled into the document type desired. Different commands create the various forms of browseable output.

5.2.1 The latex Command

The latex command compiles LaTeX into dvi format. There are many viewers for dvi files. Dvi is not as nice as postscript or PDF but does serve as an intermediate format for postscript and other formats. DVI also does not do well with color, although the postscript generated from it will. Compiling the the output from the example above would require a statement like this.

```
latex test
```

LaTeX will create several files, all of which have different purposes. The .aux files contain measurement information. For this reason it is a good idea to run the latex command twice. The second time it will use the information it gathered the first time. The result will be better looking output. The output file from this command will be test.dvi.

5.2.2 The dvips Command

The dvi2ps command converts dvi files to postscript. The resulting postscript often looks better than the dvi output. Particularly when color is used. The following dvips command will print the contents of test.dvi to your default printer. The second command will cause the postscript to be written to a file, test.ps. There are many more options to dvips, Printer, papertype, print controls, cropping, copies, to name just a few.

```
dvips test.dvi
dvips test.dvi -o test.ps
```

5.2.3 The pdflatex Command

The pdflatex command compiles LaTeX code into PDF. This works very well and makes beautiful PDF output. The pdflatex command is used in place of the regular latex command to create pdf directly from the original LaTeX output. Like the latex command, it is a good idea to run the pdflatex twice so that measurements will be refined and used. The following command will create a file named test.pdf.

```
pdflatex test
pdflatex test
```

5.3 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.

5.3.1 The easy way

The easiest thing to do is write an external stylesheet, then add a usepackage statement to your LaTeX document. All of the SAS specific needs will be in that one stylesheet file. This will provide the best support for color and formatting of the output. All of the latex macros defined in that stylesheet are prefixed with 'sas' so the macro names should not clash with any latex code you already have. Using our test example from above all you need is the following line in your preamble. You can then include any parts of the ODS generated report in your own document.

```
\usepackage[color]{test}
```

5.3.2 Using NewFile to advantage

The newfile option, along with no_top and no_bot can create nicely packaged pieces of output that can be directly included into a LaTeX document. Frequently, all that is desired is the actual tabular output generated by ODS. There are several options that will help narrow the ODS output to just the parts you need. First the select and exclude statements can be used to select only the particular ODS output objects of interest. Then the ODS Markup options, newfile, no_top_matter and no_bottom_matter can be used together to create just the tables or graphs, with no surrounding latex code. The ods statement to do that will look like this.

```
ods latex file="test.tex"(notop, nobot)
          stylesheet="test.sty"(url="test")
          newfile=table;
```

The result will be a series of numbered files, starting with test.tex. All with one table per file. These files will be much easier to include into another latex document. In fact, depending on the occurance of titles and notes, these files could be left intact and included with latex's include statement, like this.

```
\include{test}
```

5.3.3 The simple way

A more simplistic method is to use the simple_latex tagset. That tagset requires nothing outside of the most basic LaTeX. This latex has a much simpler table model that uses the tabular package, and does not support color. In short, this tagset uses only the most basic latex commands to create tables. This latex tagset does not need a stylesheet so including it's output requires no usepackage to be added in the preamble of your latex document.

5.4 Image Formats and Graph

Everything would be great if the image formats attribute actually worked. But it doesn't. At least not for graph procedures. It does work for statistical graphics though. The problem for graph procedures is that no image types other than those valid for HTML are allowed. That means that png will work but not postscript. PdfLateX likes png images, but latex and dvips do not.

The fix for this is a simple tagset that converts image file extensions from .gif to .ps. Then after the job is done, the images can be replayed to the postscript device. The following example does just that.

Listing 5.1: A fix for LaTeXand graph

```
 proc template;
    define tagset tagsets.mylatex;
       parent = tagsets.latex;

       image_formats = 'ps,psepsf,png,jpeg,gif';

        define event image;
            put '\sasgraph{';
            put BASENAME / if !exists(NOBASE);

            /* convert gif extension to ps.         */
            /* use eps if you use the psepsf driver. */
            put tranwrd(URL, 'gif', 'ps');

            put '}' CR;
        end;
    end;
 run;

filename junk ".";

ods tagsets.mylatex file="graph.tex";
goptions dev=gif target=ps;

proc gchart data=sashelp.class;
vbar age / pattid=midpoint;
run;
quit;

proc gplot data=sashelp.class;
plot height*weight;
```

```
run;
quit;

ods _all_ close;

/*------------------------------------------------------------eric-*/
/*-- Replay the graphs to generate postscript.                 --*/
/*---------------------------------------------------------16Oct03-*/
goptions dev=ps gsfname=junk;
proc greplay nofs;
   igout work.gseg;
   replay _all_;
run;
quit;
```

5.5 LaTeX in the different versions of SAS

5.5.1 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. The Embedded_stylesheet option was not an available feature in ODS Markup at in that release. An external stylesheet had to be used.

But when a stylesheet was specified, a crash occured. Because of tagsets, SAS was able to work around this problem by making a new latex tagset available. The new tagset was called latex2. The latex2 tagset solved the problem by writing a fixed set of style definitions to the preamble of the LaTeX document. No stylesheet option was needed. The disadvantage was that the ods markup style option had no effect. Using this latex in other documents was also very problematic.

5.5.2 SAS 9.0

In SAS 9.0 LaTeX worked much better. The LaTeX code itself was much cleaner and more adaptable. A common complaint was using the output as inclusions in other documents. One of the changes was the addition of the 'sas' prefix on all of the ODS generated latex macros.

5.5.3 SAS 9.1 and beyond.

LaTeX has continued to evolve with the addition of the simple Latex tagset. The latex output is more versatile than ever.

Another addition in SAS 9.2 is measured markup destination. The measured markup destination adds measurement variables to tagsets. This destination is more or less invisible, since the behavior is turned on by the tagset's measurement attribute. This is the same sort of measurement that the ODS RTF and Printer destinations do. The first goal of this destination is support an RTF tagset. But LaTeX will also benefit from this.

Turning on measurement will enable the LaTeX tagset to make intelligent decisions about table panelling and paging.

5.6 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 and papers to books. It can be used to create many output types including PDF, Postscript, and DVI.

Part II

Beginning Tagsets

Part III

Intermediate Examples

Part IV

Usage Notes and Caveat’s

Part V

Appendices

Quick Reference Guide

.1 Useful tagsets

HTML4 *Shortcut Alias: HTML, HTML4*

Generates W3C compliant HTML4.

PHTML *Shortcut Alias: pHTML*

HTML with a simplified Stylesheet.

HTMLCSS *Shortcut Alias: HTMLcss*

HTML 4.0 with a complete stylesheet definition. Only slightly different HTML from the HTML4 tagset.

XHTML This is an XHTML destination that is almost identical to HTML4.

CSV *Shortcut Alias: CSV*

CSV that contains only tabular data.

CSVALL *Shortcut Alias: CSVall*

CSV with titles, footnotes, notes, and bylines.

CSVbyline CSV with tables and bylines.

RTF This is an RTF destination with both vertical and horizontal measurement among other things. The RTF tagset is the future of ODS RTF.

LaTeX *Shortcut Alias: Latex*

The basic LateX destination for creating complete LaTeX documents.

ColorLaTeX The basic LateX destination with the color package added.

SimpleLaTeX A simple LateX destination without SAS specific LaTeX macros.

TablesOnlyLaTeX A very simple LateX destination that only generates tables. This is perfect for use with newfile=output to create files with individual tables for inclusion in other LaTeX documents.

Troff *Shortcut Alias: Troff*

This is a basic troff tagset that is useful for creating black and white typeset documents, in pdf and postscript. Troff is used to typeset the man pages on Unix systems.

MSOffice2k *Shortcut Alias: MSOffice2k*

An HTML tagset with Microsoft specific XML embedded for better loading into Microsoft Office products.

cHTML *Shortcut Alias: cHTML*

Compact HTML creates very compact HTML with no style. Used primarily for PDA's and phones.

Imode *Shortcut Alias: Imode*

Imode is a subset of HTML that has no tabular support. Used for PDA's and phones in Japan and some other countries.

WML *Shortcut Alias: WML*

Wireless markup language is a small footprint XML for PDA's and phones.

WMLolist *Shortcut Alias: WMLolist*

Wireless markup tagset that creates a table of contents as an option list.

Default *Shortcut Alias: XML*

An XML definition that is closely modeled after the internals of ODS.

Event_map Event map creates an XML file that shows all events as XML tags. Event map prints a large number of attributes as part of it's output.

Short_map Short map is a much less verbose mapping tagset that can be much easier to read. This is handy for showing the event model when looking for events, names, labels and values.

Super_map Super map is a mapping tagset that can be controlled through options. It can be verbose, or not. Among other things it can look for specific values within the event model and show only events that match.

Style_Popup Style popup is a child of HTML4 it creates HTML with a popup window that displays ODS style information for all the HTML elements on the page.

Style_Display Style Display is a child of Style Popup that creates a sample page that contains examples of all of the different style elements that ODS regularly defines.

OdsStyle The Ods style tagset writes a proc template program to the stylesheet file. The program generated is a style template with no inheritance that makes it easier to see all the attributes in use for any given element. It is not a good way to keep a style template because of maintainence issues.

Pyx *Shortcut Alias: Pyx*

Pyx is a very simple markup language that lists every single item on a separate line. Pyx is very easy to parse, grep and mold into other shapes including XML.

.2 Tagset attributes

Map *string*

Map is a list of special characters that need to be translated in order to work with the target markup language. A typical value for HTML or XML might be "&<>'"

MapSub *string* Mapsub is a list of strings that will replace the characters listed in the map. The strings separater is the first character in the list. A typical value for HTML is "/&amp;/&lt;/&gt;/&apos;/".

Split *string*

The value of slit is what the tagset will use in place of split characters when they are encountered. The typical value for HTML is "<Br>".

NoBreakSpace *string*

The value of NoBreakSpace is what the tagset will use for non-breaking spaces. The default is a regular space. HTML generally uses "&nbsp;".

Default_Event *string*

This the name of an event to use when the ODS requested event is not defined within the tagset.

Output_Type *string*

Output type is the type of output being created. HTML, LaTeX, XML, CSV, etc. This is primarily used by the SAS results window so that it can display the output correctly.

Log_Note *string*

Log Note is a note that will be printed to the log every time the tagset is used. Usage notes, release date and revision number are good things to put in the lognote.

Trademark *string*

Trademark is the string to use in place of the SAS control code for trademark. HTML uses '&trad;'.

Registered_TM *string*

Registered_TM is the string to use in place of the SAS control code for registered trademark. HTML uses '&reg;'.

Copyright *string*

Copyright is the string to use in place of the SAS control code for copyright. HTML uses '&copy;'.

Image_Formats *string*

Image formats is a space delimited list of image formats supported by the target output type. Valid values are png, gif, jpeg, bmp, svg, java, activex, ps, psepsf, epsi. If the image device is not set to a valid image format, then the first image format in the list will be used to create the image.

BeginWellFormed *string*

EndWellFormed *string*

BeginWellFormed is a string that indicates what well formed markup for the target output begins with. All text will be checked to see if it begins and ends with the value of beginwellformed and ends with the value of endwellformed, excluding leading and trailing whitespace. If the string is matched, then the characters in the string will not be remapped by the values of map and mapsub.

Default_style *string*

Default style is the name of the default style to use when a style is not specified on the ods statement.

Body *string*

Body is the name to use for the body file when automatic packages are in use. When an automatic package is in use, the file name given on the ods statement becomes the package archive name. The body name given here becomes the body file actually created and placed within the archive.

Contents *string*

Contents is the name to use for the contents file when automatic packages are in use.

Pages *string*

Pages is the name to use for the pages file when automatic packages are in use.

Frame *string*

Frame is the name to use for the frame file when automatic packages are in use.

Stylesheet *string*

Stylesheet is the name to use for the stylesheet file when automatic packages are in use.

Code *string*

Code is the name to use for the code file when automatic packages are in use.

Data *string*

Data is the name to use for the data file when automatic packages are in use.

Body_Mimetype *string*

Body_Mimetype is the mimetype to use for all body files created.

Contents_Mimetype *string*

Contents_Mimetype is the mimetype to use for all contents files created.

Pages_Mimetype *string*

Pages_Mimetype is the mimetype to use for all pages files created.

Frame_Mimetype *string*

frame_Mimetype is the mimetype to use for all frame files created.

Stylesheet_Mimetype *string*

Stylesheet_Mimetype is the mimetype to use for all stylesheet files created.

Code_Mimetype *string*

Code_Mimetype is the mimetype to use for all code files created.

Data_Mimetype *string*

Data_Mimetype is the mimetype to use for all data files created.

Default_Mimetype *string*

Default_Mimetype is the mimetype to use for all files that do not specify a mimetype.

Indent *Integer*

The value of indent is used to determine how many spaces the Ndent and Xdent statements move the indention level.

Breaktext_Width *Integer*

Breaktext_width is the maximum **width of space** that will be considered for placement of automatic breaks in text. If the width of the space is greater than this value the text will not be broken.

Breaktext_Length *Integer*

Breaktext_length is the maximum **length of text** which will be considered for placement of automatic breaks. If the text is longer than this value then no breaks will be inserted automatically

Breaktext_Ratio *Number*

Breaktext_Ratio is the ratio of the width of space to the length of the text which is supposed to fit in it. Like the other two attributes, this attribute serves to narrow the the string and width combinations that will be considered for splitting. The text length and width must fall within this ratio before they will be considered for forced splits.

Upi *Integer*

Fontpad *Integer*

Parent *Tagset Name*

Parent is the name of a tagset to inherit events and attribute settings from.

Package *Package Template Name*

Package template name. This turns on automatic packages for the tagset. When set, the file specified on the ods statement becomes the package archive name. All output from the destination will be placed in that archive. The actual files created will depend upon the settings of the package file attributes, body, contents, pages, frame, stylesheet, code and data.

Stacked_columns *yes | no | on | off*

When set to off, stacked column events will not be sent to the tagset. Except for the Freq Procedure with cross tabular reports.

Embedded_stylesheet *yes | no | on | off*

If set to on, stylesheet events will be sent to the body file when no stylesheet file is specified on the ods statement.

Pure_style *yes | no | on | off*

Normally, all style information is surfaced in the stylesheet events and is filtered in subsequent events. Setting pure_style to yes will cause all events to receive all the style information all of the time.

Measurement *yes | no | on | off*

Turn contents measurement on. When on, the output is measured vertically and horizontally and ajusted to fit on the page. Primarily used by the RTF, LaTeX and any other markup that needs page formatting.

Ext_Graph_Instance *yes | no | on | off*

No_Byte_Order_Mark *yes | no | on | off*

If set No Byte order mark will be generated for XML files.

Hierarchical_Data *yes | no | on | off*

When hierarchical data is set to yes, The Tabulate and Freq procedures generate events in a tree view rather than a tabular view.

Uniform *yes | no | on | off*

Reference_Image *yes | no | on | off*

.3 Event attributes

STYLE *Style Element Name*

This is the name of the style element that the event should use.

Pure_Style *yes | no | on | off*

When pure style is set to yes, all style attributes will be available, they will not be filtered as if they were already defined in a stylesheet.

File *(frame | contents | style | code | body | default | pages | data)*

The file attribute determines which output file the event will write it's output to. If the file was not specified on the ods statement, the event will generate no output.

.4 Event Statements

Block *event-name <condition>*

Blocks the use of the specified event. Use the unblock command to make the event usable again.

Break *<condition>*

Break stops an event from continuing. The event is exited immediately, no statements below the break statement will be executed.

Continue *<condition>;*

When placed within an if or while, execution goes back to the top for re-evaluation.

Close *<condition>*;

Close stops the flow of output to the current stream. Output is redirected to the current output file.

Delstream *(stream-name | variable) <condition>*;

Completely deletes the specified stream. This is different from an unset which simply clears the contents of the stream.

Do *<condition>*

Starts a statement block.

Done

Ends a statement block.

Else *<condition>*;

Begins a statement block that executes if the Do it belongs to is false. This is more efficient and easier to read than the common alternative of two lines with one if negated. In the case of a Do /while the else only executes if the while is false on the first evaluation. Multiple elses may be chained by using an if.

Eval *variable-name>[(<number>|<key>)]<where-clause>*;

Eval sets the value of the variable to the return value of the where clause. The variable's type can be numeric or string depending upon the where clause. The standard SAS Language where processing syntax applies.

Flush *<condition>*;

Forces any buffered output to be written to the current output file or stream.

Iterate *variable-name <condition>*;

Creates or initializes an iterator for the given list or dictionary variable. The first value of the variable will be placed into _value_. If it is a dictionary the key will be placed in _name_.

Ndent *<condition>*;

Indents output one indentation level using the number of spaces specified by the INDENT= attribute.

Next *variable-name <condition>*;

Causes the variables iterator to increment to the next value and repopulate the variables _value_ and _name_ as appropriate.

Open *(stream-name | variable) <condition>*;

Opens the specified stream. If the stream does not exist it is created. If a variable is given, the value of that variable becomes the stream name. If a different stream is open, that stream is closed. All PUT statements that occur after the open are appended to the stream and not to the output file.

PUT | PUTL | PUTQ *(<variable> <String> <function> <nl|cr|lf>)* <conditon>*;

Writes text or variable data to an output file. In general, the syntax consists of a space-delimited list of strings, variables, new line, or data step functions, followed by an optional condition. Functions may not be nested. A string preceding a variable creates a string-value pair, Both the label string and the variable will print as long as the variable has a value. PUT is the basic statement in its simplest form. For example: put 'Beginning of output.'; PUTL adds a new line to the end of the output. This is useful when an event's output is large. PUTQ places quotes around the value in a variable. For example, the PUTQ statement putq "color=" foreground; results in the following output:

color="blue" To write a new line, use any of these new line arguments: CR, NL, or LF. All three work the same. PUT statements pair strings with variables. If a string is followed by a variable, they become a pair. If the variable has a value, then the pair becomes output. If not, then neither will be output. For example, for the following PUTQ statement, if none of the variables have a value, the output would be <table>: putq "<table" " background" background " foreground=" foreground "cellpadding=" cellpadding ">" nl;

Putlog *(<variable> <String> <function>)* <conditon>*;

Putlog works just like put except that the output is directed to the log. Putlog does not accept newlines.

Putstream *(stream-name | variable) <condition>*;

Writes the contents of the specified stream to the current output file.

Putvars *(variable-group | list | dictionary) (<variable> <String> <function> <nl|cr|lf>)* <conditon>*;

Putvars is like the put statement except that it loops through all of the variables in the variable group. Each iteration populates special variables that can be used in the format. _name_ holds the name of the variable. _value_ holds the value of the variable. This is very similar to a simple loop using iterate and next. All in one statement.

Values for variable-group are EVENT STYLE DYNAMIC MEMORY STREAM

putvars event _name_ ':' _value_ nl;

Set *variable-name <[(<number>|< key>)]> (<variable> <String> <function>)* <conditon>*;

Sets the specified variable with the value of any strings, variables, or data step functions, following the variable name. Functions may not be nested. The variable name must be a memory or stream variable or stream variable, that is, preceded by a single or double dollar sign ($ or $$). The arguments follow the same behavior and syntax as the PUT statement. SET does not accept newlines. The only limitation is that a user variable cannot be set to a stream variable. Variables can be set to themselves. It should be noted that setting a stream to itself can be inefficient and slow.

Stop *<condition>*;

Causes execution to jump to the end of the current statement block.

Trigger *event-name <START | FINISH> <condition>*;

Executes another event inline. If you are in the start section of an event, then any event triggered also runs its start section. If you are in the finish section, then the triggered event runs its finish. If a triggered event does not have start or finish sections, then it runs the statements it does have. A trigger can also explicitly ask for an event's specific section.

Unblock *event-name <condition>*;

Re-enables a previously blocked event. To disable an event, use the BLOCK statement.

Unset *(ALL | variable-name [(<number>|< key>)]) <condition>;*

Remove, delete, clear, the variable given.

Xdent *<condition>;*

Removes one indention level from the output.

.5 If Statements

All of these items are conditons that can be used anywhere you see <condition> in the reference. Conditions are separated from the statement with a /. if, when, and where can be used as desired for readability. While and breakif both give special behavior to the statements they are connected to. The test can can be any of the 5 builtin tests or it can be a where clause. The 5 builtin functions are Any, cmp, contains, exists, or a variable alone.

Statement/ **(<if | when | where | while | breakif>)** ANY | CMP | CONTAINS | EXIST | VARIABLE | WHERE CLAUSE

Any *(variable, <variable>*)*

checks a list of variables for values. If any of the variables has a value, then the condition is true and the statement executes.

BreakIf *<condition>*

When true break if executes the current line and then breaks out of the event.

Cmp *(variable | string, variable | string)*

compares a string to a variable or a list of variables for equality. Cmp is case insensitive. For example:

Contains *(variable | string, <variable | string >);*

looks inside the first string for the second string. Case sensitive.

EXIST | EXISTS *(variable, <variable>*)*

checks a variable or a list of variables to determine if a value exists for each. If all of the variables have a value, then the condition is true and the statement executes. If a variable has an empty string of length 0, then the value does not exist and the statement does not execute.

Variable

An if that consists of a single variable will be evaluated according to its type. If it is a string variable the test will be for length. If it is a numeric variable the test will be for the value. If the variable is a dictionary element ("$dict[$key]") the the result is true if the key is defined, otherwise false."

While *<condition>*

While is only valid on a Do statement. It indicates that the corresponding statement block should loop until the while becomes false.

Variables

.6 Event Variables

.6.1 508 Accessibility

All of these variables for 508 Accessiblity can be set in the table template.

Abbr Abbreviation; useful for compliance with Section 508 of the U.S. Rehabilitation Act of 1973.

Acronym Acronym

Alt Alternate description.

Caption Captions for tables.

LongDesc Long table description.

Summary Table Summary.

.6.2 Data

Name Name of the current variable or key. Used by putvars, iterate and next

Value Value of the current variable or variable element. Used by putvars, iterate and next

Dname Name of the column in the data component to associate with the current column. Set with the DATANAME= attribute in the column definition.

Label Label for the variable. Set with the LABEL= attribute in the column definition.

Name Name of the variable. Set with the VARNAME= attribute in the column definition.

Value Current value. data, header, title, byline, note, etc.

.6.3 Data Formatting

Closure Describes whether the endpoints of a format range are included or excluded, i.e. <-, -, -<, <-<, etc.

DataEncoding Encoding type for Raw value, always Base64.

DataType Picture format option.

DefWidth Format Default Width

Fill Picture format option.

FMTLang Picture format option.

Fuzz Format option.

Max Format option

Min Format option

Missing Value that indicates that no data value is stored. By default, SAS uses a single period (.) for a missing numeric value and a blank space for a missing character value. In addition, for a numeric missing value, a special missing value can be used to represent different categories of missing data by assigning the letters A - Z or an underscore.

MultiLabel Format option.

Multiplier Picture format option.

NoEdit Picture format option

No_Wrap Places a NOWRAP attribute in the tag, so that the browser doesn't insert a line break. NOWRAP is automatically added to a <TD> in HTML. This is important, for example, if the cell contents are math, because the browser might otherwise break a line after a negative sign.

NotSorted Format option

Precision Number of places to the right of the decimal.

Prefix Picture format option.

RangeEnd End value of a range in a format.

RangeStart Start value of a range in a format

Round Format option

Base64 encoding of the stored machine representation of the original value.

SASFormat SAS format used to format the value.

Scale Total number of places in the floating point number.

Type Type of data, which can be STRING, DOUBLE, CHAR, BOOL, or INT.

UnformattedType Data type before formatting.

UnformattedValue Value before formatting.

UnformmatedWidth Width before formatting.

.6.4 Event MetaData

Empty Flag to determine whether event is called as an empty tag.

Event_Name SAS Requested event name.

State Current state of the event, which is either START or FINISH.

Trigger_Name Name of the current event if in a triggered event

.6.5 Graph

Archive Points to the location of the jar files. Used by SAS/GRAPH.

ClassId Identifier used by MS Windows to instantiate active controls.

Coordinate Coordinate in a map area. Used by SAS/GRAPH.

Grseg The current graph image is a Grseg.

Image_Formats Specifies a comma-separated string of image types. The image types are the same as what SAS/GRAPH can produce, for example, GIF, JPG, PNG.

Shape Shape of the clickable map. Used by SAS/GRAPH.

.6.6 Measured

Blue

Bottom

Green

KeepN

Left

List_Index

NOCenter

Page_Columns

SectionData

Red

Right

Top

Vmerge

.6.7 Miscellaneous

Anchor Current anchor, which is the last value of the anchor tag. For example, IDX.

Data_Viewer Name of the Data Viewer, Table, Batch, Tree, Graph, Report, Print, etc

Date The Date Formatted as YYYY-MM-DD.

Dest_File Current destination file. Values include BODY, CONTENTS, PAGES, FRAME, CODE, STYLESHEET, DATA.

FirstPage Specifies that the current page is the first page of the output file.

Language Language of the current output. Currently, only set when it is an Asian language.

Output_Label Label of the current output object.

Output_Name Name of the current output object.

Output_Type Output type as specified in the tagset.

Page_Count Page count since the files were opened.

Proc_Count How many procedures have run since the files were opened.

Proc_Name Name of the current procedure.

SASLongVersion Long format of the SAS version.

SASVersion Short format of the SAS version.

Space String that the tagset uses for a nonbreaking space.

Split String that the tagset uses for line breaks.

Style Current style in use.

Style_Element Name of the current style element, Always populated when possible. Where htmlclass is only populated when using stylesheets.

Suppress_Charset The Suppress Charset Registry setting.

Time The time formatted as HH:MM:SS.

TOCLevel Table of contents level.

Total_Page_Count Page count since the ODA was opened.

Total_Proc_Count How many procedures have run since the ODA opened.

.6.8 ODS Statement

Author Author of the output. Specified from the ODS statement or is the user that is running SAS.

BaseName BASE= option as set in the ODS statement.

Body_Name Name of the body file.

Body_Title Title of body file.

Body_URL URL of the body file.

Code Used by SAS/GRAPH.

CodeBase The codebase used by SAS/GRAPH.

Code_Name Name of the code file.

Code_Title Title of code file.

Code_URL URL of the code file.

Contents_Name Name of the contents file.

Contents_Title Title of contents file.

Contents_URL URL of the contents file.

Data_Name Name of the data file.

Data_Title Title of data file.

Data_URL URL of the data file.

Encoding The real world encoding which corresponds to the encoding specified. ex. utf8, iso-8859-1.

Frame_Name Name of the frame file.

Frame_Title Title of frame file.

Frame_URL URL of the frame file.

Graph_Path_Name The graph path as given on the ODS statement

Graph_Path_URL The graph path URL.

No_Bottom Non-zero if the No_Bottom_Matter option was specified.

No_Top Non-zero if No_Top_Matter option option was specified

Operator Operator. Set from the ODS statement or is the user that is running SAS.

Pages_Name Name of the pages file.

Pages_Title Title of pages file.

Pages_URL URL of the pages file.

Path Path as set by the ODS statement.

Path_Name The Path as given on the ODS statement.

Path_URL The path URL.

Stylesheet_Name Name of the stylesheet file.

Stylesheet_Title Title of stylesheet file.

Stylesheet_URL URL of the stylesheet file.

Tagset Name of the current tagset.

Tagset_Alias Tagset alias, as given by the alias attribute on the ODS statement.

Title Title from the ODS statement.

TranTab Translation table name for character conversions.

.6.9 Table

CLabel Label for the output object in the contents file, the Results window, and the trace record. Set with the CONTENTS_LABEL= attribute in the table definition.

Colcount Number of columns in the current table.

Colend Ending column number.

Colspan Number of columns that the cell spans.

Colstart Column number for which the cell starts.

Data_Row Specifies that the current row is a data row.

First_Stacked_Value Specifies that this is the first value in a set of stacked values.

Last_Stacked_Value Specifies that this is the last value in a set of stacked values.

Is_Stacked TRUE if the Current column is a stacked column.

Row Current table row, which includes headers.

RowSpan Number of rows that the current cell spans.

Section Section of the table, which can be HEAD, BODY, or FOOT.

Width Width. Most commonly used for COLSPECS.

.6.10 Title and Note Formatting

After AFTER Specifies that the current note is an after note (see the documentation for the NOTES statement and the NOTES= option).

Before Specifies that the current note is a before note (see the documentation for the NOTES statement and the NOTES= option).

Hidden Specifies when the current object is hidden. Generally used when creating the table of contents.

In_Association Specifies inside an association.

In_Caption Specifies inside the caption part of an association.

Is_Note Specifies that the current procedure title is a note.

Is_Title Specifies that the current procedure title is a title.

.6.11 URL

NoBase NOBASE Flag to determine whether to use the value for BASE= as part of the URL. 0 uses BASE=. 1 does not use BASE=.

Target TARGET Target that is associated with the URL.

URL Fully formed URL.

.6.12 XML Libname Engine

ID

SUFFIX

XMLDataForm Specifies whether the tag for an element to contain SAS variable information (name and data) is to appear in open element or enclosed attribute format. Used by the XML LIBNAME engine.

XMLMetaData Specifies whether to generate schema-related information.

XMLSchema Specifies whether to generate schema-related information.

tag_Name Specifies a dynamic tagname defined by the data.

XMLCONTROL

XMLCDATA

XMLPARM

.7 Style Variables

.7.1 Borders

BorderBottomColor Border Color

BorderBottomStyle Border Style: Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, Outset, Hidden.

BorderBottomWidth Border Width

BorderColor Color of the border if the border is just one color.

BorderColorDark Darker color in a border that uses two colors to create a three-dimensional effect.

BorderColorLight Lighter color in a border that uses two colors to create a three-dimensional effect.

BorderLeftColor Border Color

BorderLeftStyle Border Style: Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, Outset, Hidden.

BorderLeftWidth Border Width

BorderRightColor Border Color

BorderRightStyle Border Style: Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, Outset, Hidden.

BorderRightWidth Border Width

BorderRightStyle Border Style: Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, Outset, Hidden.

BorderTopColor Border Color

BorderTopStyle Border Style: Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, Outset, Hidden.

BorderTopWidth Border Width

BorderWidth Width of the border of the table.

.7.2 Font

color Color of the text.

FontFamily Font family.

FontSize Size of the font.

FontStyle Style of the font.

FontWeight Font weight.

FontWidth Font width.

textdecoration

textindent

.7.3 Background

BackgroundColor Color of the background.

BackgroundImage Background image.

Watermark Specifies whether to make the image that is specified by BACKGROUNDIMAGE into a watermark.

BackgroundRepeat

backgroundPosition

.7.4 Frame

BodyScrollBar Specifies whether to put a scrollbar in the frame that references the body file.

BodySize Width of the frame that displays the body file in the HTML frame file.

ContentPosition Position, within the frame file, of the frames that display the contents and the page files.

ContentScrollbar Specifies whether to put a scrollbar in the frames that display the contents and the page files.

ContentSize Width of the frames in the frame file that display the contents and the page files.

FrameBorder Specifies whether to put a border around the frame for an HTML file that uses frames.

FrameBorderWidth Width of the border around the frames for an HTML file that uses frames.

FrameSpacing Width of the space between frames for HTML that uses frames.

LinkColor Color for links that have not yet been visited.

ListEntryAnchor Specifies whether to make the entry in the table of contents a link to the body file.

.7.5 Miscellaneous

FillRuleWidth Width of the fill rule.

Flyover Text to show in a tool tip for the cell.

NoBreakSpace

PostHTML HTML code to place after the table or cell.

PostImage Image to place after the table or cell.

PostText Text to place after the table or cell.

PreHTML HTML code to place before the table or cell.

PreImage Image to place before the table or cell.

PreText Text to place before the cell or table.

ProtectSpecialCharacters Specifies how less-than (<) and greater-than (>) signs and ampersands (&) are interpreted.

Height Height of the object.

Padding Amount of white space on each of the four sides of the text in a cell.

PaddingLeft

PaddingTop

PaddingBottom

PaddingRight

BorderSpacing Amount of spacing between cells.

Width Width of the object.

Rules

Frame

HrefTarget Window or frame in which to open the target of the link.

Class Name of the stylesheet class to use for the table or cell.

ContentType Value of the content type for pages that you send directly to a Web server rather than to a file.

DocType Entire doctype declaration for the HTML document.

HTMLID ID for the table or cell.

HTMLStyle Individual attributes and values for the table or cell.

Margin

MarginRight

MarginLeft

MarginTop

MarginBottom

.7.6 Graph

Description

GradientDirection

ImageStyle

MarkerSymbol

TickDisplay

DisplayOpts

Connect

CapStyle

ContrastColor

StartColor

NeutralColor

EndColor

Index

/, 87
$, 77
$$, 77
$options, 31, 33

Alias, 30
any, 26, 27, 87
Attributes
    BeginWellFormed, 69
    Breaktext_Length, 73
    Breaktext_Ratio, 73, 74
    Breaktext_Width, 73, 74
    Copyright, 71
    Default Event, 71
    Default_event, 40
    Embedded Stylesheet, 72
    Embedded_stylesheet, 112
    EndWellFormed, 69
    Ext_Graph_Instance, 74
    Heirarchical_Data, 74
    Image_formats, 218
    image_formats, 71
    indent, 41, 70
    Log_note, 73
    Map, 69
    MapSub, 69
    Measured, 71
    measurement, 73
    No_Byte_Order_Mark, 74
    nobreakspace, 70
    output_type, 71
    Parent, 69
    Pure Style, 72
    pure_style, 112, 117
    Registered Trademark, 71
    Split, 73
    split, 70
    stacked_columns, 70
    style, 115
    Trademark, 71

basics, 9
BeginWellFormed, 69
Block, 139
Body, 73
Body_mimetype, 73
Break, 88
Breaktext_Length, 73
Breaktext_Ratio, 73, 74
Breaktext_Width, 73, 74
Byline
    modification, 175, 205
    Modifying, 167, 197
ByLines, 177, 207
Bylines, 165, 195

CHTML, 226
cHTML, 222
Close, 80
cmp, 26, 27, 87
Code, 73
Code_mimetype, 73
color LaTeX, 217
compare, 26, 27, 87
concatenation, 17
contains, 26, 87
Contents, 73
Contents_mimetype, 73
Continue, 90
control
    procedural, 87
        break, 87
        breakif, 87
        do, 87
        done, 87

        else, 87
Copyright, 71
Counting Observations, 165, 167, 195, 197
CSV, 25, 33, 165, 195, 225
    Delimiter, 33, 34
csv, 7

Data, 73
Data Step
    functions, 101
Data_mimetype, 73
DataStep
    Functions, 154, 184
DataStep Conversions, 165, 195
DDE, 165, 195, 226
Debugging, 180, 210
Default Event, 41, 71
Default_event, 40
Default_mimetype, 73
Define
    event, 16, 28
    tagset, 16
DelStream, 80
Destinations
    cHTML, 222
    HTML, 221
    HTML3, 221
    HTML4, 221
    HTMLCSS, 222
    MSOffice2k, 222
    PHTML, 221
    XHTML, 222
Dictionaries, 79, 91
Dictionary, 177, 207
Do, 89
Done, 89
DVI, 220
dvips, 216

Else, 89, 130
Embedded styles, 114
Embedded Stylesheet, 72
Embedded_Stylesheet, 114
Embedded_stylesheet, 112
EndWellFormed, 69
Error, 21
Eval, 101
Event
    Branch, 60
    byline, 21
    Data, 18
    data, 16, 28, 50
    error, 21
    fatal, 21
    finish:, 19
    Header, 18
    header, 17
    Hyperlink, 48, 49
    Initialize, 29
    note, 21
    proc_title, 21
    Put_all, 44
    Row, 18
    start:, 19
    style_class, 112
    system_footer, 21
    system_title, 21
    Table, 18
    Table_body, 18
    table_body, 20
    Table_foot, 18
    table_foot, 20
    Table_head, 18
    table_head, 20
    warning, 21
Event model, 15
Event Requests, 15
Events
    Bygroup, 177, 207
    Byline, 175, 177, 205, 207
    classAlign, 127
    doc, 36
    doc_body, 36, 145
    doc_title, 36
    Finding, 35, 48
    Finish:, 36, 97
    Footer_setup_container, 137, 139
    Header, 130
    Identify, 48
    Initialize, 36, 40, 154, 158, 184, 188
        Timing, 158, 188
    Nobs_label, 175, 205
    page_setup, 137

  pagebreak, 139
  Proc, 158, 188
  Start:, 36, 97
  state, 97
  Style_class, 114, 115
  System_Footer, 139
  System_Footer_setup, 137, 139
  System_Footer_setup_group, 137, 139
  System_Title, 139
  System_Title_setup, 137, 139
  System_Title_setup_group, 137, 139
  table_head, 123, 125, 173, 203
  Title_container, 139
  Title_container_row, 139
  Title_container_spec, 139
  Title_container_specs, 139
  Title_setup_container, 137, 139
  Triggering, 97
Excel
  CHTML, 226
  CSV, 225
  DDE, 226
  ExcelXP, 226
  HTML, 226
  MSOffice2k, 226
  PHTML, 226
  SpreadSheetML, 226
  SYLK, 225
ExcelXP, 226
exists, 26, 27, 87
Ext_Graph_Instance, 74

Fatal, 21
File Redirection, 55
Files
  Stylesheet, 158, 188
Finding Events, 25, 35
Finding Variables, 43
Footers, 21
Footnotes, 21, 137, 141, 145
Frame, 73
Frame_mimetype, 73
Functions, 101
  fget, 104, 154, 184
  filename, 104, 154, 184
  fopen, 104, 154, 184
  fread, 104, 154, 184
  getoption, 106
  index, 101
  input, 102
  inputn, 102
  lowcase, 106
  missing, 104
  prxmatch, 106
  prxparse, 106
  prxposn, 106
  putc, 102
  Return Code, 154, 184
  scan, 102
  strip, 101
  substr, 101
  tranwrd, 106
  trim, 101

GML, 5
groff, 6

Heirarchical_Data, 74
Highlighting, 165, 195
HTML, 226
  Class, 127
  Justification, 127
  Stylesheets, 222
html, 6
HTMLCSS, 222
htmltags.tpl, 48

if, 26, 87, 89, 90
  any, 26, 87
  Breakif, 89
  cmp, 26, 87
  compare, 26, 87
  contains, 26, 87
  exists, 26, 87
  Where clauses, 88
  While, 90
image_formats, 71
Images
  eps, 218
  png, 218
  ps, 218
Indent, 41
indent, 70

inheritance, 114, 115, 117
itemstore, 9
Iterate, 91

LaTeX, 73, 130
    SAS 8.2, 219
    SAS 9.0, 219
    SAS 9.1, 219
Latex, 216
    Color, 215
    Compiling, 216
    DVI, 216
    PDF, 217
    Postscript, 216
latex, 6
Lists, 78, 91
    Negative indexing, 78
    Popping values, 78
    Pushing values, 78
Log_note, 73
Looping, 90, 91

Macro Variables, 29
Map, 69
MapSub, 69
Measured Destination, 219
Measured Output, 219
Measurement, 71
measurement, 73
Mimetypes, 73
MSOffice2k, 222, 226

ndent, 41, 70
newline, 17
Next, 91
No_Byte_Order_Mark, 74
nobreakspace, 70
nobs, 165, 169, 195, 199
Non-breaking spaces, 70
Note, 21
Number of Observations, 169, 199
Numbers, 79
    Converting, 102
Numeric Variables, 79

Observation Count, 169, 199
ODS, 165, 195
    Document, 165, 195
    Exclude, 165, 195
    Select, 165, 195
    Styles, 165, 195
ods markup statement, 11
    Multiple Active, 12
    permutations, 12
ODS Options
    Newfile, 217
    No_bot, 217
    No_bottom_matter, 217
    No_top, 217
    No_top_matter, 217
ods statement, 11
ODS Statements
    Latex, 215
Odsxrpcs, 43
Open, 80
Options, 31, 33
    CSV, 33
    CSVALL, 33, 34
    No_Bot, 154, 184
    No_Bottom_Matter, 154, 184
    No_Top, 154, 184
    No_Top_Matter, 154, 184
    Set, 33, 34
output_type, 71

Packages, 73
    file names, 73
    Mimetypes, 73
Page breaks, 137
page breaks, 141
Pages, 73
Pages_mimetype, 73
papersize, 106
Parent, 69
PDF, 217, 220
pdflatex, 217
Perl
    Regular Expressions, 106
PHTML, 221, 226
plain_text, 16
Postscript, 216, 220
proc template, 9
procedural, 15

Procedure
    Report, 50
    report, 51
Procedure titles, 21
Procedures
    Report, 165, 195
Pure Style, 72
pure_style, 112
Put, 17, 101
Putl, 153, 183
Putlog, 46, 180, 210
Putq, 40
PutStream, 80
Putvars, 43, 81, 180, 210

Registered Trademark, 71
roff, 6
RTF, 73

SAS Registry
    suppress_charset, 40
sashelp, 9
sasuser, 9
Set, 29, 33, 34, 77, 101
sgml, 6
Shortcut destinations, 12
show_charset, 40
Simple names, 12
Split, 73
split, 70
SpreadSheetML, 226
stacked_columns, 70
    Proc Freq, 70
StartPage, 137
Statements
    Block, 139
    Break, 88
    Close, 80
    Continue, 90
    DelStream, 80
    Do, 89
    Done, 89
    Else, 89, 130
    Eval, 101–104, 106, 154, 184
    Iterate, 91
    ndent, 41, 70
    Next, 91
    Open, 80
        SAS 9.2, 80
    Put, 17, 101
        label/value pairs, 40
    Putl, 153, 183
    Putlog, 46, 180, 210
    Putq, 40
        Macro variable bug, 52
    PutStream, 80
    Putvars, 43, 81, 180, 210
        Put_all event, 44
    Set, 29, 33, 34, 50, 77, 89, 101–104, 106, 154, 184
    Stop, 90
    Trigger, 40, 97
        finish, 98
        start, 98
    UnBlock, 139
    UnSet, 77
    xdent, 41, 70
Stop, 90
Stream Variables, 80
Streams, 80, 137
    SAS 9.2, 80
Strings
    Converting to number, 102
Style, 16
    Attributes
        HTMLClass, 132
        TagAttr, 132
    table_head, 173, 203
Styles, 111, 121
    data, 126
    dataEmphasis, 126
    dataStrong, 126
    Minimal, 167, 169, 197, 199
    table_head, 125
Stylesheet, 73, 114
Stylesheet_mimetype, 73
suppress_charset, 40
SYLK, 225
sylk, 7

Table
    body, 20
    borders, 121

- cell, 16, 17
- foot, 20, 21
- head, 20
- row, 19
- rules, 121
- Striped, 126, 130
- style, 121
- thead, 123

Table of Contents, 60
- Flattening, 60

Tables
- striping, 167, 197

Tagset
- inheritance, 25, 114, 115, 117
- modifying, 25
- OdsStyle, 55

tagset_alias, 30

Tagsets
- Color_Latex, 215
- Event_map, 48
- event_map, 35
- HTML, 221
- HTML4, 48, 49, 123
- html4, 127
- HTMLCSS, 123
- HTMLcss, 48, 49
- htmlcss, 127
- Measured, 219
- MVSHTML, 49
- Odsxrpcs, 43
- PHTML, 48, 49, 123
- short_map, 35, 158, 188
- Simple_Latex, 218
- StartPage, 137
- text_map, 35
- tpl_style_map, 35

tagsets
- HTMLcss, 50

tagsets.default, 11

tex, 6

thead, 123

Titles, 21, 137, 141

Trademark, 71

Trigger, 40
- finish, 98
- start, 98

Triggers
- Example, 98

troff, 6

UnBlock, 139

UnSet, 77

usepackage, 217

Value labels, 40

Variables
- $options, 31
- colstart, 50
- Event_name, 40
- Filename, 62, 64
- Finding, 43
- Fixed Length, 104
- HTMLclass, 119
- Macro, 169, 175, 181, 199, 205, 211
- No_Bot, 154, 184
- No_Top, 154, 158, 184, 188
- Numeric, 79
- Section, 126
- State, 97
- Stream, 80
- style_element, 119
- tagset_alias, 30
- target_column, 50
- target_value, 50
- Trigger_name, 97

variables, 15
- colstart, 27
- MetaData, 15
- Style, 16

Warning, 21

Website Integration, 153, 162, 183, 192

Website Templates, 153, 183

when, 87

where, 87

Where clauses, 88

While, 90, 91

xdent, 41, 70

XHTML, 222

xhtml, 6

XML, 165, 195

xml, 6