

XML REPORTING

1

Overview	2
What is XSL?	3
Custom Reports	4
Summary Reports	8
Modifying Colors	13
Changing the Fonts	15
Outputting Positive Values in Red	16
Example of Modified XSL	19
Character Substitutions	20

OVERVIEW

The SPXML library introduces a rich report generation system for creating custom reports in a wide array of document formats. In this technical paper, we use the S-PLUS data frame `fuel.frame` to create report examples that are output in Portable Document Format (PDF), HTML, PostScript, and Rich Text Format (RTF).

This reporting system uses the Extensible Stylesheet Language (XSL) to create PDF, HTML, PostScript, and RTF reports from an XML description of an S-PLUS object. This allows a maximum degree of freedom for specifying what is included/excluded, and the manner of presentation. It also largely removes dependencies of a specific report format, so a report created in RTF can easily be created in PDF.

The SPXML library contains a small number of powerful functions that are used to generate these reports. Two functions provide the XML reporting infrastructure:

- `createXMLFile` Writes S-PLUS objects to a file in XML.
- `javaXMLTransform` Transforms XML files with XSLT or XSL-FO.

These functions are all you need to create a file with an XML description of an S-PLUS object, and generate a report from the information in this XML file. The formatting information is specified in an XSL file.

One reason for keeping the S-PLUS functions simple and focusing on XSL as a transformation mechanism is that XSL is an industry standard with a wealth of documentation on its usage. This provides you with extensive reference materials that are not included as part of the S-PLUS documentation set.

This document focuses on the use of XML as a reporting tool. Using XML for data exchange is discussed in the separate *XML Generation* technical paper, which can be accessed by navigating to **Help ► Technical Papers**.

While these report generation tools are quite useful for creating custom reports, the SPXML library also includes a `summaryReport` function that provides a specific set of summary statistics by groups. This function combined with the corresponding XSL files provide examples of how you can create sophisticated reports.

WHAT IS XSL?

XSL is a specification for transforming XML to other types of XML or to other markup formats. XSL can be broken into two parts:

- XSLT, used for transforms.
- XSL Formatting Objects (XSL-FO).

XSL transforms are used to convert an XML document from one series of element types to another. The resulting document is typically XML or HTML.

XSL formatting objects describe how to use a set of XML elements to create a formatted document with elements such as titles, sections, tables, and page breaks. The same XSL description can be used to create PDF, PostScript, or RTF files. The translation from a conceptual tag such as `<fo:table-header>` to the appropriate PDF description is generated automatically, so you don't need to know anything about the markup characters used by a specific file format.

The tags used for XSL-FO are a superset of those used for XSLT. Thus, we typically have one XSL file describing the HTML generation, and then a second XSL file with additional formatting objects tags for the other report formats.

There are several XSL files included in the **library/SPXML/xml** directory in S-PLUS:

SplusObjects.xml A simple transformation file that creates a formatted HTML version of the following S-PLUS objects: data frames, lists, arrays, vectors, matrices, functions, call objects, and named objects.

SplusObjects_FO.xml A simple transformation file that creates a formatted PDF, PostScript or RTF version of the following S-PLUS objects: data frames, lists, arrays, vectors, matrices, functions, call objects, and named objects.

ColumnReport.xml A detailed transformation file that creates an HTML report in conjunction with the S-PLUS function `summaryReport`.

ColumnReport_FO.xml A detailed transformation file that creates a PDF, PostScript or RTF report in conjunction with the S-PLUS function `summaryReport`.

CUSTOM REPORTS

To create a report from an S-PLUS object, we save the object as XML and then transform the XML with XSL. For example, to create simple reports of the data frame `fuel.frame`:

```
> library(SFXML)
> xsltFile <- paste(getenv("SHOME"),
+   "/library/SPXML/xml/SpplusObjects.xsl", sep="")
> xmlFile <- "temp.xml"
> splusObject <- fuel.frame
> createXMLFile(splusObject, xmlFile)
> javaXMLTransform(xmlFile, "fuelReport.htm", xsltFile)
```

Note

The following command would yield the same results since the default XSL file is **library/SPXML/xml/SpplusObjects.xsl**.

```
> javaXMLTransform(xmlFile, "fuelReport.htm")
```

By default, the report is created in your working directory.

In this example, **SpplusObjects.xsl** is used as the XSL transform. This is a generic transformation that can be used for data frames, lists, arrays, vectors, matrices, functions, call objects, and named objects.

In the example above, you could substitute a list, array, vector, matrix, function, call object, or name for the data frame. For example, any of the following work:

```
> splusObject <- hist
> splusObject <- list(c(1:100), fuel.frame, hist)
> splusObject <- c("A", "AA", "AAA", "AAAA")
```

In order to create a report in PDF, PostScript, or RTF, use **SpplusObjects_FO.xsl**:

```
> foFile <- paste(getenv("SHOME"),
+   "/library/SPXML/xml/SpplusObjects_FO.xsl", sep="")
> javaXMLTransform(xmlFile, "fuelReport.pdf", foFile)
> javaXMLTransform(xmlFile, "fuelReport.rtf", foFile)
> javaXMLTransform(xmlFile, "fuelReport.ps", foFile)
```

Note

If no XSL file is given, the **library/SPXML/xml/SplusObjects_FO.xsl** file is used by default for output file names with extensions **.pdf**, **.rtf** and **.ps**.

To customize the formatting of the report, make a copy of the XSL files and modify them appropriately. Examples of modifying XSL are presented in the section Summary Reports.

While we specify the XSL file explicitly in these examples, you do not need to specify the XSL file if you wish to use one of the standard XSL files. If no XSL file is specified, **SplusObjects.xsl** is used for HTML or XML transforms, and **SplusObjects_FO.xsl** is used for other types of transforms.

It is also possible to create your own XSL file for reporting purposes. As a simple example, suppose you wanted to create an HTML table from an S-PLUS matrix. To do this, you must first output the desired matrix to an XML file:

```
> splusObject <- format(cor(state.x77))
> xmlFile <- "corXMLFile.xml"
> createXMLFile(splusObject, xmlFile)
```

Then, you must create an XSL file capable of transforming the XML into an HTML table. The following is an example of such an XSL file; the entire file is located in your **library/SPXML/examples/xml_reporting/CorrelationMatrix.xsl**:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="S-PLUS">
    <html>
      <!-- create title for html page -->
      <title>Correlation Matrix for state.x77</title>

      <body>
        <!-- create a title for the top of the page -->
        <h2>Correlation Matrix for:</h2>
        <!-- create html table version of matrix -->
        <xsl:call-template name="Matrix">
          <xsl:with-param name="element" select="./Matrix"/>
        </xsl:call-template>
      </body>
    </html>
  </xsl:template>

  <xsl:template name="Matrix">
```

```

<xsl:param name="element" select=""/>

<div>
  <!-- create table for display of data frame or of matrix -->
  <table cellspacing="1" cellpadding="5" border="1">
    <!-- create a row of column headers -->
    <tr>
      <th></th>
      <xsl:for-each select="$element/Columns/Column">
        <th><xsl:value-of select="@name"/></th>
      </xsl:for-each>
    </tr>

    <!-- create an html row for each row of data in the matrix -->
    <xsl:for-each select="$element/RowNames/Items/Item">
      <xsl:variable name="rowNumber" select="position()"/>
      <tr>
        <!-- first item is row header that contains row name -->
        <th><xsl:value-of select="./text()"/></th>

        <!-- add actual data...watch for factors and missing values -->
        <xsl:for-each select="./../Columns/Column">
          <xsl:variable name="colType" select="./@type"/>
          <xsl:variable name="value" select="./Items/Item[$rowNumber]/text()"/>
          <xsl:variable name="levels" select="./Attrs/Attr[@name = '.Label']"/>

          <td align="right">
            <xsl:choose>
              <!-- character value -->
              <xsl:when test="$colType= 'character'">
                <xsl:value-of select="$value"/>
              </xsl:when>
              <!-- factor value -->
              <xsl:when test="$colType= 'factor'">
                <xsl:value-of select="$levels/Vector/Items/Item[position() = $value]/text()"/>
              </xsl:when>
              <!-- non-factor value -->
              <xsl:when test="count($levels) = 0">
                <xsl:value-of select="$value"/>
              </xsl:when>
              <!-- missing factor -->
              <xsl:when test="$value = 'NA'">
                <xsl:text/>
              </xsl:when>
            </xsl:choose>
          </td>
        </xsl:for-each>
      </tr>
    </xsl:for-each>
  </table>
</div>
</xsl:template>

</xsl:stylesheet>

```

To convert the XML file into an HTML table, use these commands:

```
> xmlFile <- paste(getenv("SHOME"),
+ "/library/SPXML/examples/xml_reporting",
+ "/CorrelationMatrix.xml", sep="")
> htmlFile <- "CorrelationTable.html"
> xml2html(xmlFile, htmlFile, xmlFile)
```

The HTML output file should look like Figure 1.1:

Correlation Matrix for:

```
createXMLFile(format(cor(state.x77)), "corXMLFile.xml")
```

	Population	Income	Illiteracy	Life Exp	Murder	HS Grad	Frost	Area
Population	1.00000000	0.20822756	0.10762237	-0.06805195	0.34364275	-0.09848975	-0.33215245	0.02254384
Income	0.20822756	1.00000000	-0.43707519	0.34025534	-0.23007761	0.61993232	0.22628218	0.36331544
Illiteracy	0.10762237	-0.43707519	1.00000000	-0.58847793	0.70297520	-0.65718861	-0.67194697	0.07726113
Life Exp	-0.06805195	0.34025534	-0.58847793	1.00000000	-0.78084575	0.58221620	0.26206801	-0.10733194
Murder	0.34364275	-0.23007761	0.70297520	-0.78084575	1.00000000	-0.48797102	-0.53888344	0.22839021
HS Grad	-0.09848975	0.61993232	-0.65718861	0.58221620	-0.48797102	1.00000000	0.36677970	0.33354187
Frost	-0.33215245	0.22628218	-0.67194697	0.26206801	-0.53888344	0.36677970	1.00000000	0.05922910
Area	0.02254384	0.36331544	0.07726113	-0.10733194	0.22839021	0.33354187	0.05922910	1.00000000

Figure 1.1: *CorrelationTable.html*, the generated HTML table.

SUMMARY REPORTS

The `summaryReport` function provides summary statistics by groups for data frame objects. Use the `args` function to list the arguments:

```
> args(summaryReport)
function(data, file, variables = c(), grouping.variables =
c(), mean = T, median = T, stdev = T, range = T, quartile =
T, st.error = T, missing = T, precision = 2, nbins = 5,
minLevels = 5, type = NULL, xslFile= NULL, title = "",
maxColumnsPerTable = 0, loggingLevel = 4)
```

These arguments are described in Table 1.1.

Table 1.1: *Arguments for summaryReport.*

Argument	Required/ Optional	Description
<code>data</code>	Required	Data frame object or set of columns
<code>file</code>	Required	Generated report's output location
<code>variables</code>	Optional	Specification for desired report columns. This allows specific columns to be targeted for report. If unspecified, the default is to include all non-grouping columns in data.
<code>grouping.variables</code>	Optional	Grouping specification. This allows the report's statistics to be grouped or broken up by certain variable values.
<code>mean</code>	Optional	Option for numeric mean
<code>median</code>	Optional	Option for numeric median
<code>stdev</code>	Optional	Option for numeric standard deviation

Table 1.1: Arguments for `summaryReport`. (Continued)

Argument	Required/ Optional	Description
<code>range</code>	Optional	Option for numeric range
<code>quartile</code>	Optional	Option for numeric quartile
<code>st.error</code>	Optional	Option for numeric standard error
<code>missing</code>	Optional	Option for numeric missing count
<code>precision</code>	Optional	Option to specify output precision
<code>nbins</code>	Optional	Specify bin count for numeric columns in groups argument
<code>minLevels</code>	Optional	Specify minimum number of unique values in numeric grouping column
<code>type</code>	Optional	Specify format: “ xml ”, “ pdf ”, “ rtf ”, “ ps ”, or “ html ”. If this argument is not specified, the extension of the filename is used to determine the output format if possible.
<code>xslFile</code>	Optional	Specify user XSL file for report generation. The default XSL used is ColumnReport.xsl (for HTML) and ColumnReport_FO.xsl (for PDF, PostScript, and RTF).

Table 1.1: Arguments for `summaryReport`. (Continued)

Argument	Required/ Optional	Description
<code>title</code>	Optional	Specify a title to appear at the top of the report.
<code>maxColumnsPerTable</code>	Optional	Specify the maximum number of data columns per table. This controls the overall width of the report by breaking wide tables into several smaller tables.
<code>loggingLevel</code>	Optional	Specify the desired level of logging during XSL transform.

To create a very simple report of a `data.frame` object:

```
> reportFilename <- "fuel.report.html"
> summaryReport(fuel.frame, reportFilename)
```

This creates a report that looks like Figure 1.2:

		Statistic
Weight	mean	2,900.83
	median	2,885.00
	stdev	495.87
	range	(1,845.00 , 3,855.00)
	quartile	(2,571.25 , 3,231.25)
	st.error	64.02
	missing	0 (0.00 %)
Disp.	mean	152.05
	median	144.50
	stdev	54.16
	range	(73.00 , 305.00)

Figure 1.2: *The output of `fuel.report.html`*

You can create a more complicated report; for example, a PDF report of the `fuel.frame` variable `Weight` grouped by `Type` is generated by running the following code:

```
> summaryReport(data=fuel.frame,
+   file="fuel.report.pdf",
+   variables=c("Weight"),
+   grouping.variables=c("Type"))
```

This creates a report that looks like Figure 1.3:

Grouping Description
Number of Categories:
Type : 6 categories

Type	Value
Compact	15
Large	3
Medium	13
Small	13
Sporty	9
Van	7

Column Summary

Weight		Type			
		Compact	Large	Medium	Small
	mean	2,821.00	3,676.67	3,195.77	2,257.69
	median	2,780.00	3,850.00	3,200.00	2,295.00
	stdev	168.92	304.56	264.01	203.00
	range	(2,575.00 , 3,110.00)	(3,325.00 , 3,855.00)	(2,765.00 , 3,610.00)	(1,845.00 , 2,580.00)
	quartile	(2,662.50 , 2,927.50)	(3,587.50 , 3,852.50)	(2,975.00 , 3,450.00)	(2,260.00 , 2,350.00)
	st.error	43.61	175.84	73.22	56.00
	missing	0 (0.00 %)	0 (0.00 %)	0 (0.00 %)	0 (0.00 %)

Figure 1.3: The PDF output in *fuel.report.pdf*.

The `summaryReport` function generates a summary of each column in the input. For greater detail, the output can be segmented by grouping information supplied as an argument. The column summary depends on column type:

- **Categorical columns** Outputs a count and percentage for each category.
- **Numeric columns** Outputs mean, range, median, quartile, standard deviation, error, and missing (as appropriate).

The default behavior is to output all information possible, but each of the numeric items can be eliminated from the output by setting the appropriate argument to F (for example, eliminate range by setting the argument `range=F`).

`summaryReport` makes use of the **ColumnReport.xml** (for HTML reports) and **ColumnReport_FO.xml** (for PDF, PostScript, and RTF reports) using XSL transforms. These files are located in the **library/SPXML/xml** directory of the S-PLUS installation.

To modify the report, you can either modify these files or make copies of the files and specify the XSL file name when using `summaryReport`.

In the following examples, we show how you can modify the XSL to customize reports:

- Change the table colors.
- Change the font used in the table cells.
- Use a different color for positive numbers in the table.

Modifying Colors

To modify the XSL used by `summaryReport`, first make copies of the **ColumnReport.xsl** and **ColumnReport_FO.xsl** files in the **library/SPXML/xml** directory of your S-PLUS installation.

If you would prefer to follow this example in an already modified XSL file, you can use **EditedColumnReport.xsl** and **EditedColumnReport_FO.xsl** in the **library/SPXML/examples/xml_reporting** directory.

The beginning of the XSL file defines various table color and font attribute sets. To change colors and fonts, you need only change the values in these definitions.

There are four (4) colors used to create the tables:

- **g_tableBG** The background color of the table.
- **g_headerBG** The background color of the table headers (row and column).
- **g_rowBG_1, g_rowBG_2** The alternating background colors (each row).

These colors are located on line 5 of the XSL transform files, and by default, the following lists their values:

```
<xsl:variable name="g_tableBG" select="#82C0FF" />
<xsl:variable name="g_headerBG" select="#CCE6FF" />
<xsl:variable name="g_rowBG_1" select="#EEEEEE" />
<xsl:variable name="g_rowBG_2" select="#FFFFFF" />
```

Run the `summaryReport` command. The default report looks like the output in Figure 1.4:

Column Summary

		V1	V2
variety	V1	12 (100.00 %)	0 (0.00 %)
	V2	0 (0.00 %)	12 (100.00 %)
	V3	0 (0.00 %)	0 (0.00 %)
	V4	0 (0.00 %)	0 (0.00 %)
	V5	0 (0.00 %)	0 (0.00 %)
	V6	0 (0.00 %)	0 (0.00 %)
	V7	0 (0.00 %)	0 (0.00 %)
	V8	0 (0.00 %)	0 (0.00 %)

Figure 1.4: *The default report HTML file.*

In this example, the table has a background color of blue, each header is a lighter variation of blue, and the rows alternate between white and light grey.

The values stored are standard hexcode versions of RGB. The first two characters (for example in `g_tableBG: 82`) represent the red component, the second two (for example in `g_tableBG: C0`) represent the green component and the last two (for example in `g_tableBG: FF`) represent the blue component.

A simple S-PLUS function can be created to convert RGB values into hexcode:

```
char2hex <- function(x) {
  if (x < 0 || x > 255) stop("x must be between 0 and 255")
  a <- floor(x/16)
  b <- x-16*a
  if (a>9) a <- LETTERS[a-9]
  if (b>9) b <- LETTERS[b-9]
  paste(as.character(a), as.character(b), sep="")
}
```

To change these colors, adjust the RGB hex code to your desired color or use the specified HTML color tags (such as **RED**, **BLUE**, **YELLOW**, **GREEN**, etc.). For example, changing the lines in **ColumnReport.xsl** to the following:

```
<xsl:variable name="g_tableBG" select="yellow"/>
<xsl:variable name="g_headerBG" select="silver"/>
<xsl:variable name="g_rowBG_1" select="#0099cc"/>
<xsl:variable name="g_rowBG_2" select="#00cccc"/>
```

and the `summaryReport` command produces a report (Figure 1.5).

Column Summary

		V1	V2
treatment	T1	3 (25.00 %)	3 (25.00 %)
	T2	3 (25.00 %)	3 (25.00 %)
	T3	3 (25.00 %)	3 (25.00 %)
	T4	3 (25.00 %)	3 (25.00 %)
reps	1	4 (33.33 %)	4 (33.33 %)
	2	4 (33.33 %)	4 (33.33 %)
	3	4 (33.33 %)	4 (33.33 %)
plants	mean	24.67	26.83
	median	12.50	16.50
	stdev	25.27	23.96
	range	(6.00 , 70.00)	(4.00 , 77.00)

Figure 1.5: Customizing the table colors of the output HTML file.

Changing the Fonts

The font for several regions can be modified in the XSL (lines 13-32 in **ColumnReport.xsl** and lines 12-43 in **ColumnReport_FO.xsl**):

- `title` Controls the major section title font.
- `subtitle` Controls the minor section title font.
- `bold-emph` Controls the minor section title font.
- `small-cap` Controls the minor section title font.
- `table-content` Controls all the table data font.
- `row-header` Controls the table row header font.
- `column-header` Controls the table column header font.

For example, to set the table data font to Courier in an HTML report, you need to add the line

```
<xsl:attribute name="face">Courier</xsl:attribute>
```

to **ColumnReport.xml** within the `table-content` attribute-set that starts on line 21. Running the `summaryReport` command creates a report like Figure 1.6:

2	4 (33.33 %)	4 (33.33 %)
3	4 (33.33 %)	4 (33.33 %)
mean	24.67	26.83
median	12.50	16.50
stdev	25.27	23.96

Figure 1.6: *Changing the fonts in the HTML output table.*

To make a similar change to the PDF, PostScript, and RTF reports, add the line

```
<xsl:attribute name="font-family">Courier</xsl:attribute>
```

to **ColumnReport_FO.xls** within the `table-content` attribute-set that starts on line 37.

Outputting Positive Values in Red

Changing fonts and colors for the whole table can be done by changing the definitions at the top of the XSL file. We may be interested in more specific formatting, such as using a different color for the positive numeric values. We can do this by editing the XSL in the places where we format the table cell values.

Because the summary report is a sophisticated report, it has a complex XSL file. The numbers in the table cells are formatted in two ways:

- Numeric data, such as mean, median, or range
- Categorical data, such as counts and percentages)

The two templates that output table data are **OutputCountAndPercent** (line 692) and **OutputNumericItem** (line 573). Two modifications must be made accordingly:

1. Modify **OutputNumericItem** by substituting the following:

```
<xsl:element name="font">
  <xsl:if test="/text() &gt; 0">
    <xsl:attribute name="color">RED</xsl:attribute>
  </xsl:if>

  <xsl:call-template name="roundNumber">
    <xsl:with-param name="string" select="/text()"/>
    <xsl:with-param name="decimalPattern" select="$decimalPattern"/>
    <xsl:with-param name="scientificPattern" select="$scientificPattern"/>
  </xsl:call-template>
</xsl:element>
```

for this:

```
<xsl:call-template name="roundNumber">
  <xsl:with-param name="string" select="/text()"/>
  <xsl:with-param name="decimalPattern" select="$decimalPattern"/>
  <xsl:with-param name="scientificPattern" select="$scientificPattern"/>
</xsl:call-template>
```

2. Modify **OutputCountAndPercent**:

```
<xsl:template name="OutputCountAndPercent">
  <xsl:param name="count" select=""" />
  <xsl:param name="percent" select=""" />
  <xsl:param name="decimalPattern" select=""" />
  <xsl:param name="scientificPattern" select=""" />
  <font xsl:use-attribute-sets="table-content">
    <xsl:element name="font">
      <xsl:if test="$count &gt; 0">
        <xsl:attribute name="color">RED</xsl:attribute>
      </xsl:if>

      <xsl:value-of select="$count"/>
      <xsl:if test="boolean($percent)">
        (<xsl:call-template name="roundNumber">
          <xsl:with-param name="string" select="$percent"/>
          <xsl:with-param name="decimalPattern" select="$decimalPattern"/>
          <xsl:with-param name="scientificPattern" select="$scientificPattern"/>
        </xsl:call-template>%)
      </xsl:if>
    </xsl:element>
  </font>
</xsl:template>
```

- Using these modifications and running the `summaryReport` command produces a report like Figure 1.7:

Column Summary

		V1	V2
variety	V1	12 (100.00%)	0 (0.00%)
	V2	0 (0.00%)	12 (100.00%)
	V3	0 (0.00%)	0 (0.00%)
	V4	0 (0.00%)	0 (0.00%)
	V5	0 (0.00%)	0 (0.00%)
	V6	0 (0.00%)	0 (0.00%)
	V7	0 (0.00%)	0 (0.00%)
	V8	0 (0.00%)	0 (0.00%)
treatment	T1	3 (25.00%)	3 (25.00%)
	T2	3 (25.00%)	3 (25.00%)
	T3	3 (25.00%)	3 (25.00%)
	T4	3 (25.00%)	3 (25.00%)
reps	1	4 (33.33%)	4 (33.33%)
	2	4 (33.33%)	4 (33.33%)
	3	4 (33.33%)	4 (33.33%)
	mean	24.67	26.83
	median	12.50	16.50
	stdev	25.27	23.96

Figure 1.7: *Outputting positive values in red.*

To modify the PDF, PostScript, or RTF reports, similar modifications have to be made to the same templates, **OutputCountAndPercent** (line 789) and **OutputNumericItem** (line 678). In this case, there are two differences between the modifications described above and what needs to be done to **ColumnReport_FO.xsl**:

- Instead of creating a font element, we need to create an `fo:inline` element.
- XSL-FO doesn't recognize the color names in this situation. Instead of RED, we must use the color code `#FF0000`.

Thus, the modified template, **OutputCountAndPercent**, looks like this:

```
<xsl:template name="OutputCountAndPercent">
  <xsl:param name="count" select="" />
  <xsl:param name="percent" select="" />
  <xsl:param name="decimalPattern" select="" />
  <xsl:param name="scientificPattern" select="" />

  <xsl:element name="fo:inline">
    <xsl:if test="$count > 0">
      <xsl:attribute name="color">#FF0000</xsl:attribute>
    </xsl:if>

    <xsl:value-of select="$count"/>
    <xsl:if test="boolean($percent)">
      (<xsl:call-template name="roundNumber">
        <xsl:with-param name="string" select="$percent"/>
        <xsl:with-param name="decimalPattern" select="$decimalPattern"/>
        <xsl:with-param name="scientificPattern"
          select="$scientificPattern"/>
        </xsl:call-template>%)
    </xsl:if>
  </xsl:element>
</xsl:template>
```

Example of Modified XSL

Example files have been included that demonstrate all the changes mentioned above:

- Modified Table Colors
- Use of Courier Font
- Using Red for Positive Numbers

For HTML reports, **EditedColumnReport.xsl** has been included in the **library/SPXML/examples/xml_reporting** directory. For PDF, RTF, and PostScript reports, **EditedColumnReport_FO.xsl** has been included.

CHARACTER SUBSTITUTIONS

Notice that in the XSL code, “greater than” is represented by `>` rather than the standard `>`. This is because certain characters are reserved by XSL and must be substituted. Table 1.2 lists the desired character and corresponding substitution

Table 1.2: *Substitutions in XML for reserved characters.*

Character	XSL Substitution
Less than (<)	<code>&lt;</code>
Greater than (>)	<code>&gt;</code>
Equals (=)	<code>=</code>
Less than or equal	<code>&lt;=</code>
Greater than or equal	<code>&gt;=</code>