The SRS Pagination Suite

User Guide

Version	Date	Comment
0.1	Jan 2006	Draft for review by beta testers

Insight Statistical Consulting Ltd
Tile Barn
High Haden Road
Glatton
HUNTINGDON PE28 5RU
Great Britain

Tel: +44 (0) 1487 830838
Email: srs@isc-ltd.co.uk
Web: http://www.isc-ltd.co.uk/software/srspagination

1 Introduction

1.1 Overview

The SRSPagination suite is a utility to assist in the accurate pagination of SAS output. It consists of two parts: the SRSPaginator is a Java utility that calculates the amount of vertical space required to print a given text string in a given font and the SRSFormatter is a Windows executable that moves titles and footnotes from the header and footer sections to the body of an RTF file created by SAS. Optionally, it will convert the RTF file to a Microsoft Word document or Adobe PDF file. The SRSPaginator can be used when using any of the SAS printable output destinations: RTF, PDF, PostScript, and even HTML or a user-written destination. The SRSFormatter, however, requires an RTF file as input.

Both the SRSPaginator and the SRSFormatter are easily incorporated into SAS code: in SAS 9, the SRSPaginator can be called directly using the data step **`JAVAOBJ`** construct in SAS 9, or by spawning a sub-process that calls the command line interface to the main paginator class. The SRSFormatter can be called from within SAS by spawning a sub process using the **`X`** statement.

As it is written in Java, the SRSPaginator will function on any platform for which Java version 1.4.2 is available. The SRSFormatter, being a Windows executable, requires a 32-bit Microsoft Windows operating system such as Windows 98, 2000 or XP.

Both the SRSPaginator and the SRSFormatter can be called on a remote server or run on the local machine.

Example programs and utility macros are included as part of the installation.

1.2 System requirements

The entire SRSPagination suite requires SAS version 8.2 or later.

The SRSPaginator requires Java Runtime Environment (JRE) 1.4.2 or later when using the **`paginatorCLI`** class to access the actual **`Paginator`** class through a command line interface or spawned process. JRE 1.4.2 is required to call the **`Paginator`** class directly using the **`JAVAOBJ`** construct in SAS 9: later versions of the JRE are not supported. Note also that **`JAVAOBJ`** is an experimental feature of SAS: correct functionality is not guaranteed.

The SRSFormatter requires a Windows 32-bit operating system such as Windows 98, ME, 2000 or XP. The SRSFormatter has been tested with Micorsoft Word 2000 and Word 2003. it should work with any version of Word 2000 and later.

**** Need to check on MS Office installation requirements ****.

1.3 Typical workflow

All material copyright © 2006 Insight Statistical Consulting Ltd

The SRS Pagination Suite can easily be incorporated into existing workflows. A typical scenario would run along the following lines

1. Prepare your data for presentation in the normal way.
2. Call the SRSPaginator to calculate the amount of vertical space required to render each observation in your dataset.
3. Create a variable to indicate the physical page on which each observation is to appear
4. Create the RTF output file in the normal way. (If using the **`REPORT`** procedure, include your page variable as a non-printing group variable with an appropriate **`COMPUTE AFTER`** block.)
5. Call the SRSFormatter to reposition titles and footers.

Notes

Steps 2 and 3 are easily combined: either in a single data step using the **`JAVAOBJ`** facility or by using the **`%paginateCLI`** utility macro included with the SRS Pagination Suite.

1.4 Changes from previous version

This is a new document. In future, the main changes from the previous version will be summarised here and will be indicated by change bars in the outside margin of each page.

2 Installation and Registration

Install utilities are provided for both Microsoft Windows and Unix/Linux. Detailed installation instructions are given in the accompanying Installation Guide, which you will find in the zip file downloaded from the ISC website.

Regardless of the operating system used to run the SRS Pagination Suite, you will be required to register your installation with Insight Statistical Consulting Ltd (ISC). This is done by means of an email sent to ISC during the installation process. You will be able to view the contents of the email before it is sent if you so wish. You will not be able to use the SRS Pagination Suite until you have registered your installation with ISC.

3 Using the SRSPaginator

The SRSPaginator can be run in three ways:

1. Directly from the command line
2. By spawning a sub process from a SAS job using the **`X`** command
3. By instantiating a data step **`JAVAOBJ`** object with the main **`Paginator`** Java class

The first two options are functionally equivalent and will be discussed together. Using the **`Paginator`** Java class via the data step **`JAVAOBJ`** construct is in many ways

All material copyright © 2006 Insight Statistical Consulting Ltd

simpler than using the CLI, but as the **JAVAOBJ** statement is only experimental in SAS 9.1.3, it cannot be recommended for production jobs at the present time.

There is no need to alter the value of the system **CLASSPATH** variable to use the SRS Pagination Suite: wrapper programs are provide for both Windows and Unix operating systems that create the correct local environment without altering any global settings.

3.1 Forced line breaks, superscripts, subscripts and ODS inline formatting

The SRS Paginator can interpret a limited number SAS inline formatting commands, line break requests and other special characters.

Whilst the SRS Pagination Suite can, in theory be used to correctly paginate RTF, PDF and PostScript files, it can only interpret native formatting commands for RTF files. The other formats are too complicated for a cell-based utility to handle correctly. Nevertheless, the SRSFormatter can be used to convert RTF files to PDF or PostScript in some circumstances. (See ***** for details.)

When using the ODS, the handling of special characters and other formatting requests is controlled by the setting of the ODS **PROCTECTSPECIALCHARS** style attribute: setting **PROTECTSPECIALCHARS=ON**, the default, means that special characters prevented from being interpreted by the output destination as control words. When **PROTECTSPECIALCHARS=OFF**, the reverse is true. The same is true with the **Paginator** and **PaginatorCLI** classes. When accessing the **Paginator** class directly, using the **JAVAOBJ** data step construct, value of **PROTECTSPECIALCHARS** can be altered on a cell-by-cell (or column-by-column) basis using the **setProtectSpecialChars()** method. When using the **PaginatorCLI** class, **PROTECTSPECIALCHARS** is set for the entire report using the **protectspecialchars** attribute of the **filedef** element and at the cell level using the **protectspecialchars** attribute of the corresponding **cell** element. These attributes can be accessed using the **defaultprotectspecialchars** and **protectspecialchars** parameters of the **%paginateCLI** macro. However, the **protectspecialchars** parameter of the **%paginateCLI** macro acts on an entire column rather than an individual cell: to control the value of protectspecialchars at the cell level when using the **PaginatorCLI** class, the txtfile will need to be created using custom code.

3.1.1 Forced line breaks

Line breaks can be forced in RTF files by emitting the **\line** or **\par** control words. **\line** emits a line break, **\par** emits a paragraph mark. For this reason, it is better to use the **\line** control word with the SRS pagination Suite: because the effect of the **\par** control word will depend on the attributes of the paragraph style in the default template, and thus may vary between installations or vary within the same installation over time. Example 5.2 demonstrates how the SRSPaginator correctly interprets the effect of the **\line** RTF control word.

All material copyright © 2006 Insight Statistical Consulting Ltd

3.1.2 Superscripts and subscripts

The ODS **ESCAPECHAR** character can be used in conjunction with the **PROTECTSPECIALCHARS** style attribute to create special effects, such as superscripts and subscripts in output files. The SRS Pagination Suite can handle superscripts and subscripts produced using ODS inline formatting. However, the RTF specification does not define how the font size is adjusted when superscripts and subscripts are emitted: it is left to the individual RTF reader to render the output as it sees fit. Therefore, the SRS Paginator cannot guarantee that its results will be accurate when measuring super- or subscripted text.

The **superScale** field of the **Paginator** class controls how the SRS Pagination Suite handles superscripts and subscripts. The **superScale** field takes values between **0** and **1**, and indicates the scaling factor by which the current font size should be altered when rendering superscripted or subscripted text. The default value is **0.67**, which appears to work well when the RTF reader is Microsoft Word. When accessing the **Paginator** class using the data step **JAVAOBJ** construct, the value of **superScale** can be altered and checked using the **setSuperScale()** and **getSuperScale()** methods. When using the **PaginatorCLI** class, the default value of the **superScale** field cannot be changed.

Example 5.4 shows how the *SRS Pagination Suite* correctlys handle superscripts and subscripts.

3.1.3 ODS inline formatting

The **Paginator** class correctly interprets common font changes induced by the **^S={}** ODS inline formatting command. Exceptions are described below. It can also correctly handle the **^{dagger}** ODS inline formatting command. Other ODS or RTF command sequences are (correctly) ignored when the **PROTECTSPECIALCHARS** style attribute is **ON**. When the **PROTECTSPECIALCHARS** style attribute is **OFF** and the **Paginator** class encounters a command that it cannot interpret, it ignores the command (that is assumes that it occupies no space in the output file) and writes a warning to the log file.

The online documentation for the SAS output Delivery System describes many font attribute settings that are supported by only a very limited number of fonts. The SRS Pagination Suite does not support these rarely used attributes. If such an attribute setting is encountered, the SRS Pagination Suite writes a warning to the log and continues processing.

Style element	Supported style attributes	Unsupported style attributes	Comments
font_face	All installed fonts		SAS supports HTML-like font names as a comma delimited list, enclosed in quotes. The SRS Pagination Suite does not: only a single font name may be specified. Enclose the name in quotes if the font name consists of

All material copyright © 2006 Insight Statistical Consulting Ltd

Style element	Supported style attributes	Unsupported style attributes	Comments
			more than one word. Quotes are optional for single word font names.
font_weight	Bold, medium	Light, demi-light, extra-light, extra-bold, demi-bold	
font_style	Roman, Italic	slant	
font_width		normal, compressed, extra-compressed, narrow, wide, expanded	Different font widths can often be obtained by specifying a different font name. For example “Arial Narrow” instead of “Arial” and font_width=narrow.

Table 1: Font attributes supported by the SRS Pagination Suite

Note: The **font=** style element is not supported by this version of the *SRS Pagination Suite*.

Example 5.5 shows how the *SRS Pagination Suite* handles font changes induced by inline formatting commands.

3.2 Using the command line interface (CLI)

The SRSPaginator command line interface is a Java class (**PaginatorCLI**) that reads an XML file containing the text to be paginated, calls the main **Paginator** Java class to derive the necessary information and then creates a second XML file containing the derived pagination information. It is the users responsibility to create the input XML file and to process the output XML file. The **%paginateCLI** utility macro supplied with the SRS Pagination Suite simplifies the process for many simple and complex situations by handling the creation of the input file, the parsing of the output file and the updating of the input dataset automatically. Novice users may prefer to read **** immediately, as the remainder of this section deals with the technicalities of using the **PaginateCLI** class directly: in most situations this is not required as the **%paginateCLI** macro will take care of the details automatically.

3.2.1 Calling the PaginateCLI Java class

The simplest form of call to the **PaginateCLI** class is as follows

```
SRSpaginateCLI.exe –in <inputfilename>
```

> *Note: On Unix/Linux systems, replace* **SRSPaginateCLI.exe** *with* **SRSPaginateCLI.sh**.
>
> *In the example programs supplied with the SRS Pagination Suite, the global macro variable* **SRSPaginateCLI** *is defined to point to* **SRSPaginateCLI.exe** *or* **SRSPaginateCLI.sh** *as appropriate. In the rest of this document, the macro variable will be sued in place of the actual file name.*

All material copyright © 2006 Insight Statistical Consulting Ltd

Additional command line parameters can be used to specify more options for the class. The commonest of these are likely to be the **-log**, and **-out** options. Full details of all the available options are given in Section 3.6 below.

When using the **paginateCLI** class from within a SAS job, simply pass the required command line as the parameter to the **X** command. For example:

```
X "&SRSPaginateCLI -in txtfile.xml -out row.xml -log mylogfile.log";
```

Of course, this assumes that **input.xml** already exists and that **output.xml** will be subsequently processed. The next few sections describe these steps in detail.

The input file that contains the text to be processed is termed the *text file*. The output file that contains information about cell and row dimensions is termed the *row file*. The following sections define the format of these files in more detail.

For the majority of cases, including those which require some special handling, the **%paginateCLI** macro will take care of technicalities of calling the **PaginateCLI** java class for you. Even in those situations where the **%paginateCLI** macro will not work, there are a number of utility macros supplied with the SRS Pagination Suite that will simplify your work. The most important of these are listed below:

%XMLFileDef	Creates the **filedef** section of the text file.
%XMLRow	Creates a **row** element of the text file. Assumes that variable names, formats and column widths exist in macro arrays named **vvar**, **fmt** and **ccol**, with upper limits defined by **nvvars** (or equivalently **nfmts** and **nccols**, as all three arrays have the same number of elements.)
%XMLCell	Creates a **cell** element of the text file.
%XMLFont	Creates a **font** element of the text file.
%XMLRowStart	Defines the start of **row** element.
%XMLRowEnd	Defines the end of a **row** element.

The following sections describe in detail the technical requirements of the input files required by and the output files created by, the **PaginateCLI** Java class.

3.3 The SRSPaginator log

The SRPaginator log file provides information about the actions performed by the SRSPaginator during a single run. The level of information can be controlled by the **-debug** command line parameter. (See Section 3.6.4 below.) By default, the log file is located in the same folder as the txtfile, and is named

<txtfile>_SRSPaginatorCLI_<date/time>_<user>_<host>.log

where

<inputfile>	is the name of the input file, minus the file extension
<date/time>	is the date and time at which the SRSPaginator began execution, in the format **yyyymmdd_hhmmss** (using an obvious notation).

All material copyright © 2006 Insight Statistical Consulting Ltd

<user> is the user name associated with the process that invoked the paginator

<host> is the host name of the computer on which the SRSPaginator was executed

This naming convention was chosen so as to minimise the chances of file naming collisions and to group all the log files associated with the processing of (different versions of) the same file to be found in a logical place and in a logical order.

The default log file name may be overridden by using the **-log** command line parameter. (See Section 3.6.3 below.) If the default log file name is changed, the user assumes responsibility for avoiding name collisions.

By default, the SRSformatter echoes the contents of the log file as it is being written to a console window. The echoed output can be suppressed by using the **-noconsole** command line parameter. (See Section 3.6.6 below.) Creation of the log file can be suppressed by using the **-nolog** command line parameter. (See Section 3.6.3 below4.5.7 below.) Naturally, use of the **-noconsole** and **-nolog** command line parameters at the same time is not recommended.

When the level of debug information requested is higher than the default level of **3**, the log can get very large indeed. The information generated for each value of **-debug** is summarised in Table 2 below. More detailed information is given in Table 3 at the end of this section.

Debug level	Information
1	Process level: licence state, input and output files.
2	Row level information
3	Cell level information
4	Context-level information: ie changes of font or inline formatting commands within cell
5	Word-by-word updates during the processing of each cell

Table 2: Summary of -debug settings for the SRS PaginatorCLI

In general, information about the state of the SRSPaginator, such as licencing information and the date and time of execution, is preceded by ****** NOTE:**, ****** WARNING:** or ****** ERROR:**. Information about the actions taken by the SRSPaginator have no prefix.

The SRSPaginator log file begins with a statement about the validity of the current SRS Pagination Suite licence:

```
**** NOTE: The licence file is valid.
**** NOTE: Licence written to 'D:\Program Files\Insight\SRS
Pagination Suite\SRSPaginator\SRSPaginator.lic'.
**** NOTE: Licence type: Evaluation (current usage is 42, permitted
usage is 100).
          The evaluation period will expire at Wed Feb 15 10:29:59
GMT 2006 (26 days from now).
```


All material copyright © 2006 Insight Statistical Consulting Ltd

If the **PaginatorCLI** class is being used as the interface to the main **Paginator** class, the SRSPaginator next records the text file it is attempting to process, together with the names and locations of the row and log files.

```
Starting to parse
d:\progra~1\insight\srspag~1\srspag~1\samples\sas\txtfile.xml at
19Jan2006 10:56:18.734
Row file is
d:\progra~1\insight\srspag~1\srspag~1\samples\sas\rowfile.xml.
Log file is
d:\progra~1\insight\srspag~1\srspag~1\samples\sas\superscript_cli_SRS
Paginator1.log.
```

Next, the SRSPaginator reports the version of Java that it is using:

```
**** NOTE: Using JVM version 1.5.0_06.
```

If the Java version detected is not 1.4.2_xx, then the SRSPaginator writes a warning to the log:

```
**** WARNING: This version of the JVM is not supported by SAS.  If
you are calling the SRSPaginator using the data step JAVAOBJ
construct, results may be unreliable.
```

This is because the only version of Java officially supported by SAS 9 is 1.4.2. However, this is only an issue if you are calling the **Paginator** class directly by using the datastep **JAVAOBJ** construct. If you are calling the **Paginator** class indirectly, by using the **PaginatorCLI** class as an interface, there is no problem.

The SRSPaginator now shows the settings detected in the **filedef** section of the text file:

```
Paginator log stream initialised.
Starting filedef processing.
  The default escape chacater is '^'.
  Default value of PROTECTSPECIALCHARS is ON.
  The default data font is Times New Roman 10pt.
Ending filedef processing.
```

At this point, the SRSPaginator is ready to start processing the actual data that will appear in the table. It begins by announcing that it is ready to start processing the first data row of the text file.

```
Starting row processing.
Start of row 1.
```

For each cell in the row, the SRSPaginator announces that it has detected the start of the cell data and reports the settings that it has detected.

```
  Cell width: 0.75in [54.0pt].
  Cell padding: 3.00pt [3.0pt].
  Available width: 0.67in [48.0pt].
  Font: Times New Roman 10pt.
```


All material copyright © 2006 Insight Statistical Consulting Ltd

Notice that the SRSPaginator deducts the width of the cell padding from the width available to the cell text. By default the cell padding is set to **3pt** to the left, right, top and bottom of the text, but may be modified by the **CELLPADDING** style attribute. A common cause of problems is forgetting to ensure that the cell padding value passed to the SRSPaginator matches the value used by the SAS style used to render the output.

On completion of processing the text contained in a cell, the SRSPaginator reports the results of its calculations.

```
  Display text [1, final]: '001-009' [width=0.46in].
  Width: 0.75in [0.67in allowing for padding.]
  Height: 17.50pt [1 lines].
  Input text: 001-009
  Font: 10pt Times New Roman
```

Both the input text and display text are reported. In straightforward cases, these two items are identical, but where the text wraps to a new line or when the input text contains inline formatting or other special characters, they will differ.

Here's an example of a cell that includes a superscript:

```
Adding cell 1...
  Cell width: 4.2cm [119.06pt].
  Cell padding: 3.00pt [3.00pt].
  Available width: 3.99cm [113.06pt].
  Font: Times New Roman 10pt.
  PROTECTSPECIALCHARS is ON.
Starting inline formatting context: ^{super.
  Display text [1, font change]: 'Here is a' [width=1.23cm].
Font is now Times New Roman 6pt.
  Context text: 'superscript and then' [width=1.80cm].
  Display text [1, font change]: 'Here is asuperscript and then'
[width=3.03cm].
Font is now Times New Roman 10pt.
Ending inline formatting context.
  Display text [1, auto split]: 'Here is asuperscript and then some '
[width=3.99cm, 'additional' is 1.41cm long].
  Display text [2, final]: 'additional normal text' [width=3.14cm].
  Width: 4.2cm [3.99cm allowing for padding].
  Height: 29.00pt [2 lines].
  Input text: Here is a^{super superscript and then} some additional
normal text
  Font: 10pt Times New Roman
Ending cell processing.
```

And here is one from a cell that wraps on many lines:

```
Adding cell 1...
  Cell width: 5.5cm [155.91pt].
  Cell padding: 3.00pt [3.00pt].
  Available width: 5.29cm [149.91pt].
  Font: Times New Roman 10pt.
  PROTECTSPECIALCHARS is OFF.
  Display text [1, forced split]: 'This is line 1.' [width=2.05cm].
  Display text [2, auto split]: 'This is a long text that wraps from
' [width=5.04cm, 'line' is 0.53cm long].
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
  Display text [3, forced split]: 'line 2 to line 3.' [width=2.22cm].
  Display text [4, final]: 'This is line 4.' [width=2.05cm].
  Width: 5.5cm [5.29cm allowing for padding].
  Height: 52.00pt [4 lines].
  Input text: This is line 1.\line This is a long text that wraps
from line 2 to line 3.\line This is line 4.
  Font: 10pt Times New Roman
Ending cell processing.
```

The SRSPaginator reports the reason for the start of a new line as forced split when the user has used an inline formatting command or other special character to request a new line; auto split it has itself determined that a new line is required and final when the last text in the cell is short enough fit on a single line. In this case, as the file was processed with **protectspecialchars** set to off, the RTF **\line** control words appear in the input text but not in the display text.

At the end of each row, the SRSPaginator reports that it has finished with the row.

```
Ending row processing.
End of row 3.
```

Finally, when the processing of the row file is complete, the SRSPaginator reports the date and time.

```
Done at 22Jan2006 18:16:33.500
```

Note that when using the evaluation version of the SRS Pagination Suite, the SRSPaginator will print messages similar to the following when the limits of the evaluation version have been exceeded.

```
**** NOTE: Functionality limited by evaluation licence.  Returning
default value from getLineHeight.
```

Naturally, the results returned by the SRSPaginator may not be accurate in these situations.

The amount of detail reported in the SRSPaginator log file is controlled by the value of the –debug parameter of the **PaginatorCLI** Java class or by the **setDebugLevel()** method when using the data step **JAVAOBJ** construct. The table below defines the level to which the debug level should be set to obtain each piece of information.

Information	Debug level	Paginator (P), PaginatorCLI(C) or Both (B)
Java version in use	Always	B
Cell available width is negative	Always	B
Failure to open log file	Always	P
Attempt to open a null log file stream	Always	B
Functionality limited by evaluation licence	Always	B
End of row details	2	B
Incorrectly specified inline formatting context	2	B
Unsupported filedef attribute	2	C
Unsupported font style/attribute	2	C
Start of row processing	3	B

All material copyright © 2006 Insight Statistical Consulting Ltd

Information	Debug level	Paginator (P), PaginatorCLI(C) or Both (B)
End of context	3	B
End of row notification	3	B
Start of cell notification	3	B
Start of cell details	3	B
Start of filedef processing	3	C
Context state (need to expand what's printed)	4	B
Forced line break	4	B
Automatic line split	4	B
Setting font for superscripting or subscripting	4	B
Start of inline formatting context	4	B
Line text at start of inline formatting context	4	B
Handled a dagger as an inline formatting context	4	B
Handled a inline style definition	4	B
Undefined inline formatting context	4	B
End of cell	4	B
End of cell details	4	B
Start of font handling	4	C

Table 3: Details of the information printed by the SRSPaginator and SRSPaginatorCLI classes

3.4 The text file

3.4.1 The purpose of the text file

The text file is an XML file that defines the input to the **`PaginatorCLI`** class. As such, it needs to provide information about both the text and format of the data to be printed in a report. Whilst SAS does provide facilities to read and write XML files, the complexity of the required structure (for example, the ability to change font and font size mid cell is required) is such that none of the facilities built into SAS can provide the necessary structure. This means that the text file must be constructed manually. The SRS Pagination Suite provides several utility macros to facilitate this task.

3.4.2 Limitations of the text file

The handling of special characters, such as RTF control words and SAS in-line formatting directives can be specified for each individual data cell. Font faces and sizes can be specified at the cell level, or even many times within the same cell.

3.4.3 The structure of the text file

The structure of the text file is defined by the **`SRSPaginator.dtd`** file in the SRS Pagination Suite install folder. The **`SRSPaginator.dtd`** file contains the following definition:

```
<?xml version='1.0' encoding='utf-8'?>
<!-- DTD for an SRSPaginator v1.0 TextFile -->

<!ELEMENT txtfile (filedef, row*, cell*) >
<!ELEMENT filedef (font)>
<!ATTLIST filedef version (1.0) "1.0"
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
                  type (RTF) "RTF"
                  escapechar CDATA  ""
                  protectspecialchars (off | on | auto) "on"
                  cellpadding CDATA "3pt"
                  returnunits (pt | cm | in | twips) "pt"
                  delimiters CDATA " -"
                  tableht CDATA ""
                  splitchar CDATA "*">
<!ELEMENT font EMPTY>
<!ATTLIST font name CDATA #REQUIRED
               size CDATA #REQUIRED
               style (plain | bold | italic | italicbold) "plain" >
<!ELEMENT row (cell*) >
<!ATTLIST row rownumber CDATA #IMPLIED >
<!ELEMENT cell (font?, text)+>
<!ATTLIST cell width CDATA #REQUIRED
               padding CDATA #IMPLIED
               delimiters CDATA #IMPLIED
               protectspecialchars (off | on |auto) "on" >
<!ELEMENT text (#PCDATA) >
```

Figure 1: The formal definition of the format of the SRSPaginatorCLI text file

Each **<!ELEMENT>** entry defines first the name of an element of the file and then the sub-elements that comprise the main element. An asterisk indicates that the sub-element may occur any number of times, including none at all. A plus sign indicates that the sub-element must occur at least once. A question mark indicates a sub-element that must occur either exactly once or not at all. **** Check this. **** So for example, the line

```
<!ELEMENT txtfile (filedef, row*, cell*) >
```

indicates that the **txtfile** element (ie the whole file) consists of a **filedef** element followed by an arbitrary number of **row** and **cell** elements. Similarly,

```
<!ELEMENT cell (font?, text)+>
```

indicates that a cell consists of an arbitrary number of **text** elements, each of which may be preceded by an optional font element.

In its simplest form, an element called **name** starts with the string **<name>** and finishes with the string **</name>**. Sandwiched between the two is the value of the element. The **text** element is defined to be

```
<!ELEMENT text (#PCDATA) >
```

which is XML-speak for free text. Thus

```
<text>This is free text, which can include numbers such as 123 and
45.678</text>
```

is a valid text element.

The definition of the font element is

All material copyright © 2006 Insight Statistical Consulting Ltd

```
<!ELEMENT font EMPTY>
```

indicating that it has no value. However, it does have attributes, as indicated by its attribute list:

```
<!ATTLIST font name CDATA #REQUIRED
               size CDATA #REQUIRED
               style (plain | bold | italic | italicbold) "plain" >
```

This says that the **font** element has three attributes: **name**, **size** and **style**. The **name** and **style** attributes are free text, but the **style** attribute must take one of the values **plain**, **bold**, **italic** or **italicbold**.

Attributes are specified as name/value pairs within the main element definition. Thus,

```
<font name="Times New Roman" size="10pt" style=plain>
```

is the **font** element that corresponds to the **data** ODS style element in the default **styles.RTF** style.

> *Note: The DTD defines that the name and font attributes of the style element are free text, but it does nothing to ensure that the values defined for these attributes are valid font names and font sizes, or that the defined fonts that are available in the current operating environment . That is the responsibility of the programmer.*

Returning to the definition of a **cell** element, it can now be seen that

```
<cell width=3cm>
  <text>This is the cell text</text>
</cell>
```

and

```
<cell width=2in cellpadding=2pt>
  <text=This is text rendered in the default font specified in  the
filedef element.</text>
  <font name="Arial" size="12pt" style=italic>
  <text>And this is text rendered in Arial 12pt Italic."</text>
</cell>
```

are both valid **cell** definitions. However,

```
<cell width=2in cellpadding=2pt>
  <text=This is text rendered in the default font specified in  the
filedef element.</text>
  <font name="Courier New" size="12pt" style=italic>
  <font name="Arial" size="12pt" style=italic>
  <text>And this is text rendered in Arial 12pt Italic."</tixt>
</cell>
```

is not – because the second **text** element is preceded by two (not one or no) **font** elements and because the terminator of the second **text** element is **</text>**, not

All material copyright © 2006 Insight Statistical Consulting Ltd

</text>. When the **PaginatorCLI** class encounters an element that does not conform to the definition provided by the DTD, it reports the error and stops executing.

This level of rigour may at first sight appear onerous, but it is required to ensure that the results returned by the **SRSPaginator** class are accurate and have not been corrupted by the inadvertently invalid user input. Moreover, using the utility macros provided as part of the *SRS Pagination Suite* will maximise the chance that the row file conforms to the required format definition.

3.5 The row file

3.5.1 The purpose of the row file

The row file provides the SAS system with the results of the calculations returned by the **Paginator** Java class. The definition of its contents is given by the **rowfile.map** file in the SRS Pagination Suite install folder. This file can be used by the SAS 9 XML engine to read the file directly into a dataset. The code to do this is extremely simple:

```
LIBNAME BaseXML XML "<rowfile>" XMLMAP="&srs_installdir.RowFile.map";
```

Where **<rowfile>** is the name of the row file to be imported. This statement creates a dataset named **BaseXML._PageData**, which can be used in exactly the same way as any other SAS dataset.

> *Note: There is a bug in SAS 9.1 (**** and other versions? ****) that causes problems when a dataset accessed by the XML engine is used as the input to a procedure. The work around is simple: just **SET** the XML dataset into a conventional SAS dataset in a data step.*

The variables in the **BaseXML._PageData** dataset are:

Variable	Type	Description
_Row	**NUM**	The row of the table to which this observation relates. This should equate to **_N_** in the input dataset if the PaginatorCLI class has performed correctly. Takes integer values from **1** to **n** where **n** is the number of observations in the dataset.
_Cell	**NUM**	The cell (variable) within the row to which this observation relates.
_RowHeight	**NUM**	The height of this row. Equal to **max(_CellHeight)** over all observations with this value of **_Row**.
_RowLines	**NUM**	The height of the row in lines. Equal to **max(_CellLines)** for all observations with this value of **_Row**.
_RowHeightUnits	**CHAR**	The units in which **_RowHeight** is measured. One of **pt**, **in**, **cm**, **twips**.
_CellLines	**NUM**	The number of lines required to render the contents of this cell
_CellHeight	**NUM**	The height of this cell
_CellHeightUnits	**NUM**	The units in which **_CellHeight** is measured. One of **pt**, **in**, **cm**, **twips**.
_Page	**NUM**	The physical page on which this observation will appear

All material copyright © 2006 Insight Statistical Consulting Ltd

**** Check this. Does _RowHeight and _RowLines correspond to the overall row height, or simply the row height so far (left to right)?

**** Does the XML engine work correctly in SAS Version 8.x?

3.5.2 The structure of the row file

**** Is a DTD for the row file required, or does the previous section provide all the required information.

3.6 Parameters of the SRSPaginator CLI

The following parameters are recognised by the SRSPaginator CLI. A parameter may be specified more than once. In this situation, the last value to be specified takes precedence. Parameters are read from left to right on the command line. CLI files (see Section 3.6.9 below) are read from top to bottom.

3.6.1 The –in <filename> parameter

Defines the text file to be processed. The filename should include the full absolute path to the file. The structure of the file is given in Section 3.4 above.

Default: **`txtfile.xml`** in the SRS Pagination Suite installation folder.

3.6.2 The –out <filename> parameter

Defines the row file to be created. The file should include the full absolute path to the file. The structure of the file is given in Section 3.5 above.

Default: **`rowfile.xml`** in the SRS Pagination Suite installation folder.

3.6.3 The –log <filename>parameter

Defines the name of the log file that will contain the summary of the work carried out by this invocation of the SRSPaginator. The default log file name is

`<infilename>_SRSPaginatorCLI_yyyymmdd_hhmmss_<user>_<host>.log`, where

`<infilename>`	is the name of the file specified by the –in parameter, shorn of its extension.
`yyyymmdd_hhmmss`	is the date and time at which the SRSPaginator created the file. The format should be obvious.
`<user>`	is the user name of the process that created the log file
`<host>`	is the name of the host on which the process that created the log file was running.

The default log file name may be slightly cumbersome, but it does minimise the chance of access conflicts in a multi-user environment.

All material copyright © 2006 Insight Statistical Consulting Ltd

Users may override the default value at will, but in doing so they assume responsibility for maintaining the uniqueness of the log file name. Overwriting an existing log file will not cause the SRSPaginator fail, but it may damage any audit trailing processes that your organisation has in place. In multi-user environments, should two invocations of the SRSPaginator attempt to access the same log file simultaneously, at least one invocation will fail.

3.6.4 The –debug <value> parameter

Specifies the level of processing information written to the log file. The higher the value, the greater the detail. Valid value are integers in the range **`1`** to **`5`** inclusive.

Default: 3

For more information on the effect of the –debug parameter on the amount of information written to the log file, see Section 3.3 above.

3.6.5 The –nolog parameter

Optional. If present, suppresses the creation of the log file.

3.6.6 The –noconsole parameter

Optional. If present, suppresses writing log file information to the console window.

3.6.7 The –licence parameter

Optional. If present, writes a summary of the current licence status at the top of the log file.

3.6.8 The –environment parameter

Optional. If present, writes an enumeration of the Java VM system properties to the head of the log file.

3.6.9 The –cli parameter

Specifies the name of a text file that contains additional parameter specifications. The format of the file is one line per parameter. This includes parameters that themselves take arguments. For example:

```
-in
txtfile.xml
```

not

```
-in txtfile.xml
```

command files can be nested by including a **`-cli`** parameter within the text file.

All material copyright © 2006 Insight Statistical Consulting Ltd

3.7 Using the JAVAOBJ construct

**** To do ****

4 Using the SRSFormatter

4.1 What the SRSFormatter does

The SRSFormatter assumes that all tables created by SAS table the following generic form:

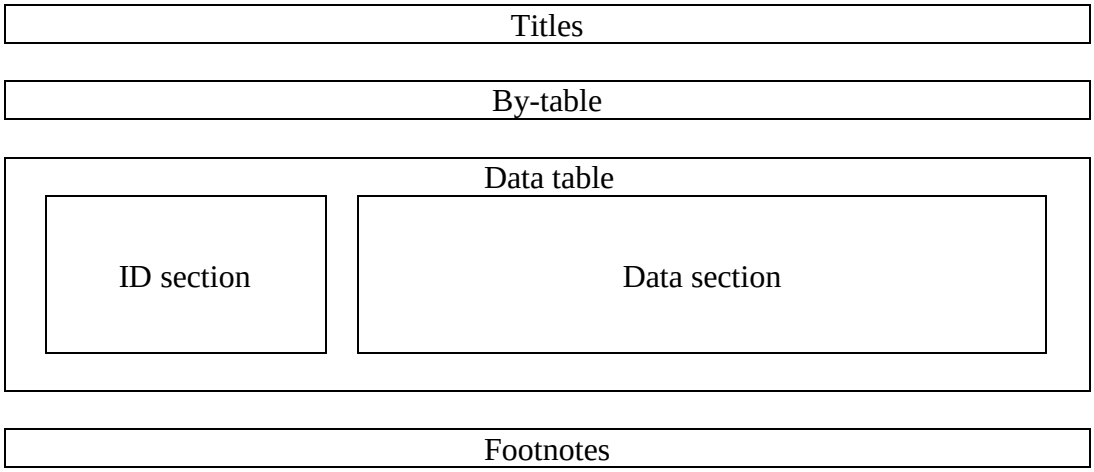

Figure 2: Generic table layout

The only required element of the table is the *data section* of the *data table.* All other elements are optional. The SRSFormatter assumes that the titles and footnotes have been created by standard SAS **`TITLE`** and **`FOOTNOTE`** statements and therefore can be found in the header and footer sections of the input RTF file.

By default, when a table or listing flows over more than one physical page in an RTF file, SAS abrogates all responsibility for the pagination of the output document. It assumes that the RTF reader, typically Microsoft Word, will handle the page layout internally. For this reason, it writes the text of any **`TITLE`** and **`FOOTNOTE`** statements that exist to the section headers of the RTF file. This meets the functional requirements of SAS, but creates many practical issues for end users. These include:

- The fact that titles and footnotes can be physically remote from the data they describe
- The problem of overwritten running titles and footnotes when SAS outputs are included as sub documents within a master document.

The SRSFormatter therefore takes any titles that it finds in the input file and moves them from the header section to the top of the first subsequent table that it finds. Similarly, it moves and footnotes to the foot of the immediately preceding table. It can also perform a number of housekeeping tasks such as resizing the titles and footnotes so that they occupy the same physical width as the data section of the table, converting section breaks to simple page breaks and converting document fields (such as the page numbers created by the **`^{page}`** and **`^{pageof}`** inline formatting commands) to plain text. These last two features are particularly useful when the

All material copyright © 2006 Insight Statistical Consulting Ltd

many RTF files are to be combined into a single master document. All of these housekeeping tasks can be turned on or off by one or more command line parameters.

In most tables, the *by-table* is not present, but when it is, the *by-table* and the *data table* form a master/detail pair. SAS has no native method of creating this kind of master/detail layout, so the SRSFormatter assumes that when a page with no footnotes is followed by a page with no titles, then a master/detail layout is being requested. The SRSFormatter then deletes the section or page break separating the two tables and replaces it with a blank line. At the same time, it processes the titles and footnotes that are present in the standard way. A more detailed explanation of the creation of master detail tables is given in Section 5.7.

All material copyright © 2006 Insight Statistical Consulting Ltd

4.2 Counting pages in Microsoft Word

RTF files created by SAS do not, by default, contain explicit page numbers, even when the inline formatting commands **thispage**, **lastpage** or **pageof** are used. (The action of each inline formatting command is described in Table 4). Instead, SAS creates document fields which are then interpreted by the RTF reader when the file is displayed. By doing this, SAS simplifies its own task by delegating the calculation of the actual page numbers to the RTF reader. Unfortunately, the particular behaviour of Microsoft Word then makes the task of the SRS pagination Suite more complicated.

In line formatting command	Action in RTF destination
thispage	Emits a **PAGE** field
lastpage	Emits a **NUMPAGES** field
pageof	Emits a **PAGE** field, the text " of " and a **NUMPAGES** field

Table 4: The effect of inline formatting commands associated with page numbering

This is because Word does not update the values of the page numbering fields automatically. Word does so only

- immediately prior to printing the document
- when the Print Preview command is issued
- when requested by the user

The first two possibilities are not available to the *SRS Pagination Suite* as they require manual intervention, and we require a solution that can run in batch. The *SRS Pagination Suite* can, and does, request that Word recalculates the page numbers in the document, but the problems do not finish there.

The next problem is that when Word repaginates a document, it does so as a background process, and there is no way of determining when the repagination has finished. Therefore, it is impossible for the *SRS Pagination Suite* to know that the values in the page numbering fields are accurate. Most of the time, this will not be a problem as the page numbers will indeed be correct, but occasionally, especially when processing large master-detail files, they will not be.

To allow for this situation, the *SRS Pagination Suite* provides three methods for determining the number of pages in the document.

By default, the value of **NUMPAGES** document field is used. Aletrnatively, each instance of the **NUMPAGES** field is replaced by the number of sections in the document. (This works because SAS, by default, emits a section break (next page) every time a page break is required.) Finally, as a last resort, the user may explicitly specify the value to be used. This behaviour is controlled by the **-pagemethod** and **-pagecount** command line parameters, which are described in sections 4.5.24 and 4.5.25 below.

4.3 The SRSFormatter log file

The SRSFormatter log file provides information about the actions performed by the SRSFormatter during a single run. The level of information can be controlled by the

All material copyright © 2006 Insight Statistical Consulting Ltd

-debug command line parameter. (See Section 4.5.5 below.) By default, the log file is located in the same folder as the input file, and is named

<input_file>_SRSFormatter_<date/time>_<user>_<host>.log

where

<inputfile>	is the name of the input file, minus the file extension
<date/time>	is the date and time at which the SRSFormatter began execution, in the format **yyyymmdd_hhmmss** (using an obvious notation).
<user>	is the user name associated with the process that invoked the formatter
<host>	is the host name of the computer on which the SRSFormatter was executed

This naming convention was chosen so as to minimise the chances of file naming collisions and to group all the log files associated with the processing of (different versions of) the same file to be found in a logical place and in a logical order.

The default log file name may be overridden by using the **-log** command line parameter. (See Section 4.5.3 below.) If the default log file name is changed, the user assumes responsibility for avoiding name collisions.

By default, the SRSformatter echoes the contents of the log file as it is being written to a console window. The echoed output can be suppressed by using the –noconsole command line parameter. (See Section 4.5.6 below.) Creation of the log file can be suppressed by using the –nolog command line parameter. (See Section 4.5.7 below.) Naturally, use of the **-noconsole** and **-nolog** command line parameters at the same time is not recommended.

In general, information about the state of the SRSFormatter, such as licencing information, the version of Microsoft Word being used and the date and time of execution, is preceded by ****** NOTE:**, ****** WARNING:** or ****** ERROR:**. Information about the actions taken by the SRSFormatter have no prefix.

An SRSFormatter log file begins with three notes that provide information about the state of the SRS Pagination Suite licence and the versions of the SRSFormatter and Microsoft Word being used.

```
**** NOTE: Licence: Timed [licence expires at 03Jan2007 09:26:07.07
(274 days from now)].
**** NOTE: Created by SRSFormat version version 1.0 [build 1].  Date
04/04/2006 07:58:35.
**** NOTE: Using Word Version 9.0 [Word 2000].
```

The SRSFormatter reports the start of each new document section as the input file is formatted:

```
Processing section 1...
```

If a master-detail relationship is detected (see Section 4.1 above), this is reported:

All material copyright © 2006 Insight Statistical Consulting Ltd

```
  Sections 1 and 2 form a master/detail pair.
```

Once titles and footnotes in all sections in the document have been processed, the SRSFormatter begins to reformat the entire document, reporting each stage as it does so:

```
Performing whole document formatting...
  Converting fields to conventional text.
  Converting section breaks to standard page breaks.
  Creating document format flag.
```

Next, the SRSFormatter saves the output file(s) in the formats requested:

```
  Saving output file(s)...
    Saving document as f:\test1.rtf.
```

Finally, the SRSFormatter reports the date and time at which processing finished, and the time taken to processed the input file:

```
**** NOTE: Finished at 04Apr2006 08:00:45.45.  Elapsed time:
00:02:10.
```

The –debug command line parameter controls the level of detail written to the log file as follows:

Value of debug	Information provided
1	Reserved for future use
2	Note detection of master-detail tables Note stages of whole document formatting Note creation of document format flag Note names of output file(s)
3	Record start of processing for each section Note conversion of section breaks to standard page breaks Note conversion of section breaks to standard page breaks
4	Program option settings, including those left at their default values Report details of field text substitution

Table 5: Values of the SRSFormatter -debug command line parameter

The default value of **-debug** is **3**. Notes, warnings and errors are always written to the log file. The value of **-debug** has no effect on the information provided by the **-fulltimer** command line option. (See Section 4.5.15.)

4.4 Using the SRSFormatter to alter a table's appearance

Although not necessary for its main task of moving **TITLE**s and **FOOTNOTE**s so that they form part of the table to which they relate, the SRSFormatter can also perform some additional formatting tasks.

The first of these, which is enabled by default, is that the rows containing **FOOTNOTE**s can be resized so that they are the same width as the table to which they are attached.

All material copyright © 2006 Insight Statistical Consulting Ltd

(By default their width is equal to distance between the left and right page margins as defined by the the SAS **OPTIONS** statement.)

This behaviour can be turned off by using the **-nofwidth** command line parameter.

More usefully, the SRSFormatter provides a basic mechanism for adding vertical separators between some columns of the table. (Prior to SAS 9.1.x, no cell specific border styling was possible. In SAS 9.1.x, directional border formatting is possible – for example, all right borders can look different to all top borders - but it is still not possible to directly create separators between some columns and not between others. The SRSFormatter gives you the option to do this in some circumstances.

For example, suppose in a table that summarises the severity of adverse events experienced in a clinical trial, you want vertical separators to appear between treatment groups, but not between the severity categories within a subgroup. This can be achieved using the **-sepcols**, **-bsepcols** or **-hsepcols** command line parameters. See Example **** for more details.

The **-*b*sepcols** command line parameter affects the *body* of the table, whereas **-*h*sepcols** affects the *header* of the table. **-sepcols** acts as if both **-bsepcols** and **-hsepcols** are specified.

The major limitation of this mechanism arises in tables with multi-line headers that also containing spannign headers. In these situations, the index of the same column will be different in different rows of the header. Because **-hsepcols** does not allow row specific indexing, this may lead to undesirable side-effects.

4.5 Command line options

Parameter names are case insensitive, so and capitalisation or other formatting is purely to enhance readability.

Only the **-in** parameter is mandatory. All others are optional, so in its simplest form, the SRSFormatter is extremely easy to use, whilst retaining the flexibility necessary for real life operation.

4.5.1 -in <filename>

Required. The full path name of the RTF file to be formatted. If the path includes white space, enclose it in double quotes.

4.5.2 -out <filename>

Optional. The full path name of file to which the formatted file is to be saved. If the path includes white space, enclose it in double quotes.

If the **-out** option is omitted, the input file is overwritten.

If the extension of the **-out** file is **.doc**, then the output file is saved as a Word document and the doc parameter need not be specified. If the extension of the **-out**

All material copyright © 2006 Insight Statistical Consulting Ltd

file is not **.doc** and the doc parameter is present, then the output file is saved as both the **-out** file (in the appropriate format) and as a Word Document. In this case, the Word document has the same path and name as the **-out** file, but the extension is changed to **.doc**.

4.5.3 -log <filename>

Optional. The name of the file to which the log for this execution of the SASFormatter will be saved. See Section 4.3 above for more information.

See also: **-nolog**, **-noconsole**, **-debug**.

4.5.4 -NoLockFields

Optional. If present, fields in the RTF file are not converted to normal text.

4.5.5 -Debug <x>

Optional. Default 2. x can take the values **1**, **2**, **3**, **4** or **5**. Specifies the level of debug information written to the log file and console window. If **-debug 1** the information written is minimal. If **-debug 4** or **-debug 5**, copious amounts of information are written. Generally, this level of detail is necessary only for debugging a program that is not functioning as expected. Values of **-debug 2** or **-debug 3** are sufficient to check the correct execution of the program in most circumstances.

Details of exactly what is written the log file for each value of debug, and how to interpret the log, can be found in Section 4.3 above.

See also: **-notimer**, **-fulltimer**.

4.5.6 -Noconsole

Optional. If present, prevents log information being written to the console window in which SASFormat is executing.

See also: **-nolog**.

4.5.7 -Nolog

Optional. If present, prevents log information being written to the log file.

See also: **-noconsole**.

4.5.8 -NoConvertSectionBreaks

Optional. If present, prevents the conversion of section breaks in the RTF file to standard page breaks.

4.5.9 -NoFWidth

All material copyright © 2006 Insight Statistical Consulting Ltd

Optional . If present, prevents the reformatting of the footnotes table so that its width matches that of the body table.

4.5.10 -NoTimer

Optional. If present, prevents simple “time started” and “time finished” information from being written to the log file and console window.

See also: **`-fulltimer`**

4.5.11 -Reformat

Optional. SRSFormat writes an invisible flag to the RTF file once reformatting is complete. By default, SRSFormat will not format for a second time a file that has already been formatted once. However, if the **`-reformat`** parameter is specified, then the file may be reformatted again.

See also: **`-noflag`**.

4.5.12 -NoFlag

Optional. If present, prevents the “formatting done” flag from being written to the RTF file.

See also: **`-reformat`**.

4.5.13 -doc

Optional. If present, saves the input RTF file in Word document format as well as the format implied by the extension of the **`-out`** file. Not required if the extension of the **`-out`** file is **`.doc`**

For example,

```
SRSFormat –in F:\test.rtf –doc
```

Processes **`F:\test.rtf`** and saves it as both **`F:\test.rtf`** and **`F:\test.doc`**.

See also: **`-rtf`**, **`-out`**, **`-in`**.

4.5.14 -rtf

Optional . If present, saves the input RTF file in RTF as well as the format implied by the extension of the **`-out`** file. Not required if the extension of the **`-out`** file is **`.rtf`**.

See also: **`-doc`**, **`-out`**, **`-in`**.

4.5.15 -FullTimer

All material copyright © 2006 Insight Statistical Consulting Ltd

Optional. If presents writes detailed timing information on the various stages of reformatting to the log file and console window. See **** for more details.

See also: **-log**, **-noconsole**, **-nolog**, **-debug**, **-notimer**.

4.5.16 -FileConverter <name>

Optional. Not yet fully implemented. If present, allows saving of the **-out** file in a format other than RTF or Word document. The value of name must be one of the file converters present in the local installation of Microsoft Word. See **** for details.

4.5.17 -Visible

Optional. If present, Microsoft Word runs in a visible window whilst reformatting the RTF file.

4.5.18 -BSepCols (<comma-separated-list>)

Optional. If present, indicates which columns in the table body are to have visible right margins. See **** for details.

See also: **-HSepCols**, **-SepCols**.

4.5.19 -HSepCols (<comma-separated-list>)

Optional. If present, indicates which columns in the table header are to have visible right margins. See **** for details.

See also: **-BSepCols**, **-SepCols**.

4.5.20 -SepCols (<comma-separated-list>)

Optional. If present, acts as if both **-BSepCols** and **-HSepCols** have been specified with the same value list. See **** for details.

See also: **-BSepCols**, **-HSepCols**.

4.5.21 -cli <filename>

Optional. If present, specifies that the file defined by **<filename>** contains additional command line parameters. **<filename>** should be an ASCII file with one command or option per line. Lines should be terminated by the default end of line sequence for the operating system on which the SRS pagination Suite is installed. **-cli** command line parameters can be nested within cli files.

Note that the format requires options to parameters to appear on the line(s) following the actual parameter. Thus,

```
-in myinfile.xml
```


All material copyright © 2006 Insight Statistical Consulting Ltd

Is incorrect. The correct syntax is

```
-in
myinfile.xml
```

4.5.22 -printer

Optional. Allows printing of the input file to a file using any of the printers installed on the local system. The extension of the output file is given by the value of the **-printertype** command line parameter. For example, the following run of SRSFormat creates a PDF file using the PDF Create utility by the ScanSoft Corporation (www.nuance.com). The name of the output file is **F:\test.pdf**.

```
SRSFormat –in F:\test.rtf –printername "ScanSoft PDF Create!"
```

4.5.23 -printertype

Optional. Default: **pdf**. Specifies the extension of the output file created by the **-printer** command line parameter.

4.5.24 -pagemethod <value>

Optional. Default **1**. Defines the method used to determine the number of pages in the formatted document. Permitted values are:

Value	Description
1	Use the current value of the individual **NUMPAGES** field
2	Use the number of sections in the document
3	Use the value supplied by the user in the **–pagecount** command line parameter.

See Section 4.2 above for more details.

See also: **-pagecount**.

4.5.25 -pagecount <value>

Optional. Sets –pagemethod to 3 and uses <value> to replace any instance of the **NUMPAGES** field found in the document.

See Section 4.2 above for more details.

See also: **-pagemethod**.

5 Examples

The code for all these examples can be found in the **/samples/sas** folder under the SRS Pagination Suite install folder. Datasets without a libref specified can similarly be found in the **/samples/data** folder. Datasets, formats and template catalogs were created using SAS 9 running on Microsoft Windows XP. Users of other versions of SAS or running on other platforms may need to recreate the input files. A transport

All material copyright © 2006 Insight Statistical Consulting Ltd

file is provided for this purpose, and the code to do so is supplied in **import_transport_files.sas**. Similarly, the code to create the necessary formats and styles can be found in **define_formats.sas** and **define_styles.sas** respectively.

The libnames required to access the data, together with the other statements necessary to create the SRS Pagination Suite environment are all contained in **autoexec.sas**. The statements are all straightforward. **autoexec.sas** should be run before any of the code in the example files. By default, this will happen automatically if the examples are run in batch using the windows explorer, but **autoexec.sas** will have to be included manually if the examples are run interactively.

**** What's the situation on Unix?

The first two examples both produce essentially the same output. The differences between them lie in the way the file is produced. Example 5.1 shows the way in which the output would normally be produced, by calling the **%paginateCLI** macro. This produces the shortest source code and hides all of the complexity of the process from the user. Example 5.2 shows how the same file can be produce d without recourse to any of the supplied macros. Although the code in example 5.2 is more complex than is necessary for this particular example, it demonstrates the techniques that need to be used in situations for which the standard **%paginateCLI** macro is inappropriate.

5.1 A simple listing using the %paginateCLI macro

Source file:	violations_listing_cli_macro.sas
Output file:	violations_listing_cli_macro.rtf
SAS log file:	violations_listing_cli_macro.log
SRS Paginator log file:	violations_listing_cli_macro_SRSPaginator.log
SRS Formatter log file:	violations_listing_cli_macro_SRSFormatter.log

The **Violations** dataset contains details of the protocol deviations noted in a clinical study. This information is to be listed. The variables containing the details of the deviation and the investigators comment are **DevDet** and **Comment** respectively. These two variables will define the amount of space required by each row of the table: the text produced by the other variables in the dataset is trivially short. The observations are to be sorted by treatment (**Drggroup**) and subject number (**PID**).

Sort the data into the required order. Although the **REPORT** procedure doesn't require its input dataset to be sorted by the **ORDER** variables, the **SRSPaginator** class does require sorted data: the **SRSPaginator** class has no knowledge of SAS formats and so cannot work out what the correct order of unsorted data would be:

```
PROC SORT DATA=DATA.Violations  OUT=Violations;
  BY Drggroup PID;
RUN;
```

Next, call the **%paginateCLI** macro. **&indata** specifies the input dataset, **&outdir** specifies the folder in which the text file, row file and log file will be created.

All material copyright © 2006 Insight Statistical Consulting Ltd

&varlist defines the variables that will appear in the report, and the order in which they do so. **&colwidths** defines the widths of the columns in the table. There should be a one-to-one correspondence between the elements of **&varlist** and those of **&colwidths**. Column widths can be specified in inches (**in**), centimeters (**cm**), points (**pt**) or twips (**twips**).

```
%paginateCLI(indata=Violations,
             outdir=&srs_sasdir,
             varlist=  Drggroup PID    Visit  DevDet Comment Major,
             colwidths=1.25in   0.75in 1.25in 2in    1.75in  0.5in)
```

That's the only additional preparatory code required to paginate the dataset. The report can now be compiled using standard **REPORT** procedure statements.

```
ODS ESCAPECHAR="^";
OPTIONS ORIENTATION=LANDSCAPE PAPERSIZE=ISO_A4;
ODS LISTING CLOSE;
ODS RTF FILE="violations_listing_cli_macro.rtf"
STYLE=styles.LandscapeA4;
PROC REPORT DATA=Violations MISSING;
  COLUMNS _Page Drggroup PID Visit DevDet Comment Major
          ("Height" _RowLines _RowHeight);
  DEFINE _Page/ORDER ORDER=INTERNAL NOPRINT;
  DEFINE Drggroup/ORDER ORDER=INTERNAL LEFT STYLE=[CELLWIDTH=3cm];
  DEFINE PID/ORDER ORDER=INTERNAL STYLE=[CELLWIDTH=1.5cm];
  DEFINE Visit/DISPLAY STYLE=[CELLWIDTH=3cm];
  DEFINE DevDet/DISPLAY STYLE=[CELLWIDTH=5cm];
  DEFINE Comment/DISPLAY STYLE=[CELLWIDTH=4cm];
  DEFINE Major/DISPLAY STYLE=[CELLWIDTH=1.5cm];
  DEFINE _RowLines/DISPLAY "Lines" STYLE=[CELLWIDTH=1.5cm];
  DEFINE _RowHeight/DISPLAY "Points" FORMAT=F5.1
                    STYLE=[CELLWIDTH=1.5cm];
  BREAK AFTER _Page/PAGE;
  TITLE1 "Listing of Violations";
  FOOTNOTE1 J=R "Page ^{pageof}";
  FOOTNOTE2 J=L ITALIC
    "Output: violations_listing_cli_macro.rtf (&sysdate9 &systime)";
RUN;
ODS RTF CLOSE;
```

The code that is specific to the operation of the SRS Paginator is indicated by *Italics*. The remainder of the code is absolutely standard **REPORT** procedure code. The code specific to the SRS Paginator will now be explained in more detail.

The **_Page** variable is a variable created by the **%paginateCLI** macro that indicates the physical page on which the observation will appear. We need to force page breaks within the report. We do this with **BREAK AFTER** statement

```
BREAK AFTER _Page/PAGE;
```

In order for the **BREAK AFTER** statement to work, the **_Page** variable must be defined as a **GROUP** or **ORDER** variable. Since we don't want any summary statistics to be produced by **_Page**, we choose to define it as an **ORDER** variable. Since we don't want the **_Page** variable to be printed in the report, we specify the **NOPRINT** option in the variable's definition.

All material copyright © 2006 Insight Statistical Consulting Ltd

```
DEFINE _Page/ORDER ORDER=INTERNAL NOPRINT;
```

And finally, the **_Page** variable must be included as the first variable in the report's columns definition:

```
COLUMNS _Page Drggroup ...
```

Optionally, the SRSFormatter can be called to reposition the reports titles and footnotes:

```
X "&srsformat -in &srs_sasdir.violations_listing_cli_macro.rtf";
```

5.2 The same listing built from scratch

Source file:	violations_listing_cli.sas
Output file:	violations_listing_cli.rtf
SAS log file:	violations_listing_cli.log
SRS Paginator log file:	violations_listing_cli_SRSPaginator.log
SRS Formatter log file:	violations_listing_cli_SRSFormatter.log

Finally, this section shows how to build the txtfile without recourse to any macro.

As before, the initial sort remains the same as in Example 5.1 above. The txtfile is constructed in a **DATA _NULL_** step. **_X** is a temporary variable used to store the formatted values of any numeric variables that will appear in the table. The txtfile is a simple ASCII file.

```
DATA _NULL_;
  LENGTH _X $ 120;
  FILE "txtfile.xml" LRECL=32767;
```

Read the observations in the **Violations** dataset. The only special requirement is the need to know when the end of the dataset has been reached.

```
SET Violations (KEEP=Drggroup PID Visit DevDet Comment Major) END=E;
```

The first section of the txtfile, as with any XML file, is the filedef section. The format of this section is rigid and the following skeleton code should be used in all cases:

```
/* filedef element */
IF _N_ EQ 1 THEN DO;
  PUT "<?xml version='1.0' encoding='utf-8'?>";
  PUT "<!DOCTYPE txtfile SYSTEM ""..\..\SRSPaginator.dtd"">";
  PUT "<txtfile>";
  PUT "   <filedef version=""1.0""";
  PUT "            type=""RTF""";
  PUT "            escapechar=""""";
  PUT "            protectspecialchars=""on""";
  PUT "            cellpadding=""3pt""";
  PUT "            returnunits=""pt""";
  PUT "            delimiters="" -""";
  PUT "            splitchar=""*""";
  PUT "            tableht=""7cm""";
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
  PUT "   >";
  PUT "      <font name=""Arial"" size=""9pt""
                style=""plain""></font>";
  PUT "   </filedef>";
END;
```

The **<filedef>** element supplies the default values of various parameters used by the SRSPaginator when paginating a table. It appears only once in the txtfile, and must appear at the head of the file. The default values it defines are valid only for the current invokation of the SRSpaginator, and can be varied within the constraints defined by the DTD shown in Section 3.4.3 above. The **version** and **type** parameters should not be altered in any circumstances.

The location of the DTD should be specified ***relative to the folder in which the txtfile is located***.

The start of a new row of data is identified by the **<row>** element:

```
PUT "   <row rownumber=""" _N_ +(-1) """>";
```

Within a row, the start of a new cell is identified by a **<cell>** element, for example:

```
PUT "    <cell width=""1.25in"" padding=""3pt"" delimiters=""- "">";
```

The cell text is defined by the **<text>** element, for example:

```
PUT "         <text>" _X +(-1) "</text>";
```

The end of a cell definition is marked by the **</cell>** element, so the code necessary to provide the SRSFormatter with the information necessary to calculate the space needed to display the **Drggroup** variable is thus:

```
_X = PUT(Drggroup, DrgLbl.);
PUT "    <cell width=""1.25in"" padding=""3pt"" delimiters=""- "">";
PUT "       <text>" _X +(-1) "</text>";
PUT "    </cell>";
```

In this case, the data value is displayed using the default font specified in the **<filedef>** element shown above. If a different font was required, this could be achieved by including a **<font>** element immediately before the **<text>** element, for example:

```
PUT "<cell width=""1.25in"" padding=""3pt"" delimiters=""- "">";
PUT "  <font name=""Courier New"" size=""10pt""
            style=""plain""></font>";
PUT "  <text>This text will be displayed in Courier New 10pt</text>";
PUT "</cell>";
```

If a font must change within a cell, this can be accommodated by including an arbitrary number of **<font>**/**<text>** element pairs. For example:

```
PUT "<cell width=""1.25in"" padding=""3pt"" delimiters=""- "">";
PUT "  <font name=""Courier New"" size=""10pt""
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
            style=""plain""></font>";
PUT "  <text>This text will be displayed in Courier New 10pt</text>";
PUT "  <font name=""Times New Roman"" size=""12pt""
            style=""plain""></font>";
PUT "  <text>and this in Times New Roman 12pt</text>";
PUT "</cell>";
```

After the cells for all the necessary variables have been defined ina similar manner, the row definition is ended by a **</row>** element.

```
PUT "   </row>";
```

Once all the observations have been written to the txtfile, the txtfile must be terminated neatly:

```
IF E THEN DO;
  PUT "</txtfile>";
  CALL SYMPUT("_N1", COMPRESS(PUT(_N_, BEST.)));
END;
```

And finally, the data step is terminated with a **run;** statement.

```
RUN;
```

The subsequent call to the **%paginateCLI** macro and the remainder of the program is identical to that shown in Example 5.1 above.

5.3 Forcing line breaks

Source file:	forced_line_breaks.sas
Output file:	forced_line_breaks.rtf
SAS log file:	forced_line_breaks.log
SRS Paginator log file:	forced_line_breaks_SRSPaginator.log
SRS Formatter log file:	Not applicable.

As explained in section 3.1.1, forced line breaks are correctly handled by the **Paginator** class when the ODS **PROTECTSPECIALCHARS** style attribute is set to **OFF**. This example demonstrates how to set the **Paginator** class up to do this.

> *Note to beta testers. I will rewrite this example once the* **PROTECTSPECIALCHARS** *option can be set at the cell level.*

Create some test data.

```
DATA Test;
  LENGTH Text $ 200;
  Text = "This is line 1. \line This is line 2. \line This is line
3.";
  OUTPUT;
  Text = "This is line 1. \line This is a long text that wraps from
line 2 to line 3. \line This is line 4.";
  OUTPUT;
  Text = "This is a wrapped text \par that uses the \\par control
word \par rather than the \\line control word.";
  OUTPUT;
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
RUN;
```

Call the **%paginateCLI** macro. Note the use of the **defaultprotectspecialchars** parameter.

```
%paginateCLI(indata=Test,
             outdir=&srs_sasdir,
             varlist=  Text,
             colwidths=6cm,
             defaultprotectspecialchars=off)
```

Create the output file.

```
ODS RTF FILE="&srs_sasdir.forced_line_breaks.rtf";
PROC REPORT DATA=Test;
  COLUMNS Text _RowLines _RowHeight;
  DEFINE Text/DISPLAY STYLE=[PROTECTSPECIALCHARS=OFF CELLWIDTH=6cm];
  DEFINE _RowLines/DISPLAY "Number of lines";
  DEFINE _RowHeight/DISPLAY FORMAT=F5.1 "Row height (points)";
  TITLE "Demonstration of the handling of forced line breaks by the
SRS Pagination Suite";
RUN;
ODS RTF CLOSE;
```

The detection and handling of the forced line breaks is illustrated by this extract from the Paginator log file:

```
Adding cell 1...
  Cell width: 5.5cm [155.90551181102362pt].
  Cell padding: 3.00pt [3.0pt].
  Available width: 5.29cm [149.90551181102362pt].
  Font: Times New Roman 10pt.
  Display text [1, forced split]: 'This is line 1. ' [width=2.15cm].
  Display text [2, auto split]: 'This is a long text that wraps from
' [width=5.04cm, 'line' is 0.53cm long].
  Display text [3, forced split]: 'line 2 to line 3. '
[width=2.33cm].
  Display text [4, final]: 'This is line 4.' [width=2.05cm].
  Width: 5.5cm [5.29cm allowing for padding.]
  Height: 52.00pt [4 lines].
  Input text: This is line 1. \line This is a long text that wraps
from line 2 to line 3. \line This is line 4.
  Font: 10pt Times New Roman
Ending cell processing.
```

Note that the RTF control word **\line** appears in the input text but not in the display text.

The output file includes an example of how the effects of using the **\line** and **\par** RTF control words can differ. Here is a section of the output file:

All material copyright © 2006 Insight Statistical Consulting Ltd

Text	Number of lines	Row height (points)
This is line 1. This is line 2. This is line 3.	3	40.5
This is line 1. This is a long text that wraps from line 2 to line 3. This is line 4.	4	52.0
This is a wrapped text that uses the \par control word rather than the \line control word.	3	40.5

Figure 3: An illustration of the different effects of the \line and \par RTF control words

Notice the difference between lines 1 and 3 of the table. Both occupy three lines, and the **`Paginator`** class reports this fact correctly. But the physical height of row 3 is clearly greater than that of row 1, even though the **`Paginator`** class reports the same height for the two rows. The reason for the discrepancy becomes apparent when the paragraph properties for row 3 are examined. Figure 4 below shows the Microsoft Word 2000 paragraph properties dialog.

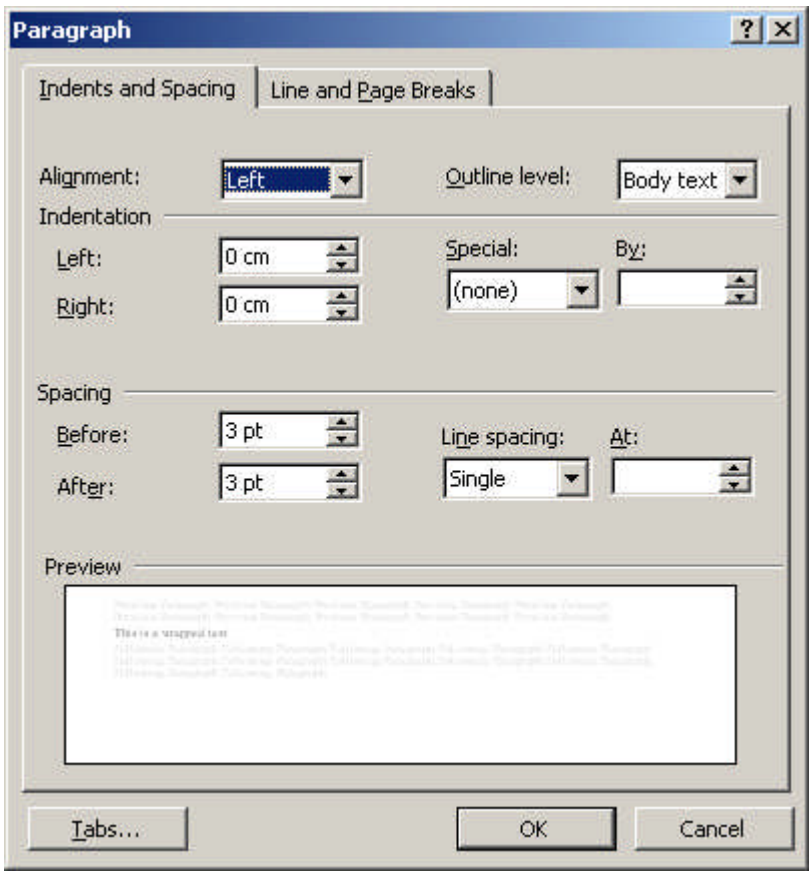

Figure 4: The Microsoft Word paragraph properties dialog for row 3

All material copyright © 2006 Insight Statistical Consulting Ltd

This clearly shows that the Normal template on this installation adds a spacing of three points above and below each paragraph. As the **Paginator** class has no way of knowing what additional styling an individual RTF reader will apply to a document, it cannot take this into account when calculating cell heights. The problem does not exist when the \line control word is used to force a line break as no additional formatting is associated with a simple line break.

5.4 Superscripts and other special characters

Source file:	superscript_cli.sas
Output file:	superscript_cli1.rtf
	superscript_cli2.rtf
SAS log file:	superscript_cli.log
SRS Paginator log file:	superscript_cli_SRSPaginator1.log
	superscript_cli_SRSPaginator2.log
SRS Formatter log file:	Not applicable.

As a simple demonstration of the handling of superscripts and superscripts, consider the following data step.

```
DATA Temp;
  LENGTH X $ 100 Y $ 100 Z $ 100;
  X = "Here is a^{super superscript and then} some additional normal
text";
  Y = "Here is a{\super superscript and then} some additional normal
text";
  Z = Y;
RUN;
```

The intention is clearly that the values of **X**, **Y** and **Z** should all be displayed as

Here is a$^{\text{superscript and then}}$ some additional normal text

However, the appearance of the value of **X** will depend on the value of the ODS **ESCAPECHAR** character and that of **Y** and **Z** on the value of the **PROTECTSPECILCHARS** attribute of the cell in which the value is displayed.

Here is a call to **PROC REPORT** that will illustrate these facts:

```
ODS ESCAPECHAR="^";
PROC REPORT DATA=Temp NOFS SPLIT="*";
  COLUMN X Y Z;
  DEFINE X/DISPLAY STYLE=[CELLWIDTH=4.2cm];
  DEFINE Y/DISPLAY STYLE=[CELLWIDTH=4.2cm];
  DEFINE Z/DISPLAY STYLE=[CELLWIDTH=4.2cm PROTECTSPECIALCHARS=OFF];
  TITLE "Handling a superscript in a column that requires wrapping";
RUN;
ODS RTF CLOSE;
```

The width of the columns has been specifically chosen so that the number of lines required for each will be different, whilst requiring both to wrap.

The call to the **%paginateCLI** macro that handles this situation correctly is

All material copyright © 2006 Insight Statistical Consulting Ltd

```
%paginateCLI(indata=Temp,
             outdir=&srs_sasdir,
             varlist=           X     Y     Z,
             protectspecialchars=on   on     off,
             colwidths=         4.2cm 4.2cm 4.2cm,
             escapechar=%str(^),

logfile=%str(&srs_sasdir.superscript_cli_SRSPaginator1.log),
             keepcelldata=TRUE)
```

Note the use of the **escapechar** and **keepcelldata** parameters. The value of the **escapechar** parameter must match that defined by the **ODS ESCAPECHAR** statement. **keepcelldata** is used to add the cell specific size data to the output dataset. In this case, this extra information is not strictly necessary, but it is kept for illustrative purposes. To display it, the original **PROC REPORT** code is modified as follows:

```
ODS ESCAPECHAR="^";
ODS RTF FILE="superscript_cli1.rtf";
PROC REPORT DATA=Temp NOFS SPLIT="*";
  COLUMN X Y Z _CellLines1 _CellLines2 _CellLines3 _RowHeight
_RowLines;
  DEFINE X/DISPLAY STYLE=[CELLWIDTH=4.2cm];
  DEFINE Y/DISPLAY STYLE=[CELLWIDTH=4.2cm];
  DEFINE Z/DISPLAY STYLE=[CELLWIDTH=4.2cm PROTECTSPECIALCHARS=OFF];
  DEFINE _CellLines1/DISPLAY "Lines required by column X";
  DEFINE _CellLines2/DISPLAY "Lines required by column Y";
  DEFINE _CellLines3/DISPLAY "Lines required by column Z";
  DEFINE _RowHeight/DISPLAY "Row height*(pt)" FORMAT=F4.1;
  DEFINE _RowLines/DISPLAY "Row lines";
  TITLE "Handling a superscript in a column that requires wrapping";
RUN;
```

This produces **superscript_cli1.rtf**, which shows that **X** and **Y** are displayed as expected. The text that was intended to be RTF control text in the variable **Y** is interpreted as simple text because the **PROTECTSPECIALCHARS** style attribute has been left at its default value of **ON** for this column. Consequently, the cell text is displayed over three lines rather than two. This fact has been correctly identified by the SRSPaginator, as in demoinstrated by the right hand columns in the table.

For completeness, **paginator_cli.sas** creates a second table that displays the same data. In this table, the cell widths are 7.5cm. In this case, **Y** should just fit on one line, but **Z** still requires two lines. This is indeed what we see, and what the SRSPaginator predicts:

Y	Z
Here is a superscript and then some additional normal text	Here is a superscript and then some additional normal text

The additional cell data is also correct, but has been omitted from the illustration for purely cosmetic purposes. The full table cane be seen in **superscript_cli2.rtf**.

All material copyright © 2006 Insight Statistical Consulting Ltd

Note that the ODS allows the **PROTECTSPECIALCHARS** style attribute to be set on a cell-by-cell basis, whereas the **ESCAPECHAR** can be set only globally.

5.5 Inline formatting

Source file:	inline_formatting.sas
Output file:	inline_formatting.rtf
SAS log file:	inline_formatting.log
SRS Paginator log file:	inline_formatting_SRSPaginator.log
SRS Formatter log file:	Not applicable.

As a simple demonstration of the handling of inline formatting, consider the following data step.

```
DATA Test;
  LENGTH X $ 200 Y $ 200;
  X = "This is Times New Roman and this is Arial 12pt bold and this
is back to Times New Roman.";
  Y = "This is Times New Roman and ^S={FONT_FACE = Arial FONT_SIZE =
12pt FONT_WEIGHT=BOLD}this is Arial 12pt bold ^S={}and this is back
to Times New Roman.";
RUN;
```

The visible text produced by printing variable X and Y will be the same. However, Y contains some ODS inline formatting commands that will alter the font used to display the text, and hence the space required to render it. The SRSPaginator should reflect this correctly.

Call the SRSPaginator. As the differences between the dimensions of individual cells are of interest, specify **keepcelldata=TRUE**.

```
%paginateCLI(indata=Test,
             outdir=&srs_sasdir,
             varlist=  X      Y,
             colwidths=4.25cm 4.25cm,
             escapechar=%str(^),
             keepcelldata=TRUE)
```

Close the listing destination and open the RTF destination.

```
ODS ESCAPECHAR="^";
ODS LISTING CLOSE;
ODS RTF FILE="&srs_sasdir.inline_formatting.rtf";
```

Create the output using the **REPORT** procedure.

```
PROC REPORT DATA=Test;
  COLUMN X Y ("Cell 1" _CellLines1 _CellHeight1)
             ("Cell 2" _CellLines2 _CellHeight2)
             ("Row"    _RowHeight _RowLines);
  DEFINE X/DISPLAY STYLE=[CELLWIDTH=4.25cm];
  DEFINE Y/DISPLAY STYLE=[CELLWIDTH=4.25cm];
  DEFINE _CellLines1/DISPLAY "Lines";
  DEFINE _CellLines2/DISPLAY "Lines";
  DEFINE _RowLines/DISPLAY "Lines";
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
  DEFINE _CellHeight1/DISPLAY "Height (pt)" FORMAT=F5.1;
  DEFINE _CellHeight2/DISPLAY "Height (pt)" FORMAT=F5.1;
  DEFINE _RowHeight/DISPLAY "Height (pt)" FORMAT=F5.1;
  TITLE1 "SRS Paginator correctly calculates text length when inline
formatting commands are present";
RUN;
```

Finally, close the RTF destination.

This code creates the following table in the RTF file.

		Cell 1		Cell 2		Row	
X	Y	Lines	Height (pt)	Lines	Height (pt)	Height (pt)	Lines
This is Times New Roman and this is Arial 12pt bold and this is back to Times New Roman.	This is Times New Roman and **this is Arial 12pt bold** and this is back to Times New Roman.	4	52.0	4	54.3	54.3	4

Although the texts in both cells require four lines to be printed, the fact that some of the text in the **Y** cell is rendered in Arial 12 pt bold means that the physical space required for this cell is greater than that required by the **X** cell. The results of the call to the SRSPaginator reflect this: the height of the **X** cell is 52pt and the height of the **Y** cell is 53.1pt. The overall row height also correct.

Also, the SRSPaginator calculates the position of the line breaks correctly. Here is an extract from the SRSPaginator log file corresponding to the **X** cell:

```
**** NOTE: Starting text processing.
  Display text [1, auto split]: 'This is Times New Roman '
[width=4.02cm, 'and' is 0.49cm long].
  Display text [2, auto split]: 'and this is Arial 12pt bold '
[width=3.88cm, 'and' is 0.49cm long].
  Display text [3, auto split]: 'and this is back to Times '
[width=3.74cm, 'New' is 0.63cm long].
**** NOTE: Ending text processing.
```

And the corresponding section corresponding to the **Y** cell:

```
 **** NOTE: Starting text processing.
  Display text [1, auto split]: 'This is Times New Roman '
[width=4.02cm, 'and' is 0.49cm long].
  Display text [2, font change]: 'and ' [width=0.60cm].
  Inline style specification detected.
    Font is now Arial bold 12pt.
  Display text [2, auto split]: 'and this is Arial 12pt '
[width=3.92cm, 'bold' is 0.63cm long].
  Display text [3, font change]: 'bold ' [width=0.95cm].
  Inline style specification detected.
    Empty inline style definition.  Resetting to default context.
    Font is now Times New Roman 10pt.
  Display text [3, auto split]: 'and this is back to Times '
[width=3.74cm, 'New' is 0.63cm long].
**** NOTE: Ending text processing.
```


All material copyright © 2006 Insight Statistical Consulting Ltd

Notice that the location of the second line break is correctly predicted in both cases.

5.6 A table that requires custom pagination

Source file:	special_processing.sas
Output file:	special_processing.rtf
SAS log file:	special_processing.log
SRS Paginator log file:	specal_processing_SRSPaginator.log
SRS Formatter log file:	Not applicable.

The **SASHELP.Shoes** dataset contains sales data from a fictitious chain of shoe shops. In this example, we build a simple summary of sales figures based on a slightly modified version of this dataset. The report should show total sales for each category of merchandise within each region. The region names should appear as spanning headers within the body of the table and should include details of the countries within the region. In addition, the information for an individual region should not span two pages. You can see the final table in the **special_processing.rtf** file.

The only minor problem here is working out how many rows are required to print the first observation in each region. (For simplicity, we assume that the table cells are sufficiently wide to allow the sales data to be rendered in a single line. If this were not the case, or if a region were permitted to span across pages, the modification to the code would be straightforward.)

First, create a format the maps the region name to its constituent countries:

```
PROC FORMAT;
  VALUE $RegFmt (MAX=120)
   “Africa"="Africa: Algeria, Angola, Egypt, Ethiopia, Kenya, South
Africa, Sudan and Uganda"
   "Asia"="Asia: Japan, South Korea and Thailand"
   "North America"="North America: Canada and The United States"
   "Central America/Caribbean"="Central America/Caribbean: Jamaica,
Mexico, Nicaragua and Puerto Rico"
   "Eastern Europe"="Eastern Europe: Czech Repiblic, Hungary, Poland
and Russia"
   "Middle East"="Middle East: Dubai, Israel and Saudi Arabia"
   "Pacific"= "Pacific: Australia, Indonesia, Malaysia, New Zealand,
Singapore and The Philippines"
   "South America"="South America: Argentina, Bolivia, Brazil, Chile,
Colombia, Uruguay and Venezuaela"
   "Western Europe"="Western Europe: Denmark, France, Germany, Great
Britain, Italy, Portugal, Spain and Switzerland";
RUN;
```

And modify the regions in the original dataset slightly

```
DATA Shoes;
  SET SASHELP.Shoes;
  IF Region IN ("United States", "Canada") THEN Region = "North
America";
RUN;
```

Create the summary dataset

All material copyright © 2006 Insight Statistical Consulting Ltd

```
PROC SQL;
  CREATE TABLE Sales AS
    SELECT Region FORMAT $RegFmt100.,
           Product,
           SUM(Stores) AS Stores,
           SUM(Sales) AS Sales,
           SUM(Inventory) AS Inventory,
           SUM(Returns) AS Returns
    FROM Shoes
    GROUP BY Region, Product
    ORDER BY Region, Product;
QUIT;
```

Call the **%paginateCLI** macro to add the pagination data to the dataset. Note that the cell width passed to the macro is equal to the sum of the widths of the columns that the headers will span.

```
%paginateCLI(indata=Sales,
             varlist=Region,
             colwidths=11.5cm,

logfile=%str(&srs_sasdir.special_processing_SRSPaginator.log))
```

Because the **%paginateCLI** macro does not make allowance for the spanning row headers, the pagination data returned by default must be slightly modified:

```
DATA Sales;
  RETAIN _Page 0 _LinesLeft 0;
  SET Sales (DROP=_Page _RowHeight _RowHeightUnits);
  BY Region;
  IF FIRST.Region AND (_LinesLeft LT (8 + _RowLines)) THEN DO;
    _Page = _Page + 1;
    _LinesLeft = 30;
  END;
  _LinesLeft = _LinesLeft - 1;
  IF FIRST.Region THEN DO;
    _LinesLeft = _LinesLeft - _RowLines;
    _RowLines = _RowLines + 1;
  END;
  ELSE _RowLines = 1;
RUN;
```

Print the full dataset to the listing file to verify that the data is correct (in this case we can't add the validation columns to the table, as this would increase the width of the table and so affect the word wrapping of the spanning headers).

```
ODS LISTING CLOSE;
ODS RTF FILE="special_processing.rtf" STYLE=Styles.LandscapeA4;
PROC REPORT DATA=Sales SPLIT="*";
  COLUMN _Page Region Product Stores Sales Inventory Returns;
  DEFINE _Page/ORDER NOPRINT;
  DEFINE Region/ORDER NOPRINT;
  DEFINE Product/DISPLAY "Product"
                 STYLE=[CELLWIDTH=3.5cm INDENT=0.5cm];
  DEFINE Stores/DISPLAY "Number*of stores" STYLE=[CELLWIDTH=2cm];
  DEFINE Sales/DISPLAY "Total sales" FORMAT=DOLLAR10.
              STYLE=[CELLWIDTH=2cm];
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
  DEFINE Inventory/DISPLAY "Inventory*value" FORMAT=DOLLAR10.
                  STYLE=[CELLWIDTH=2cm];
  DEFINE Returns/DISPLAY "Returns*value" FORMAT=DOLLAR10.
                 STYLE=[CELLWIDTH=2cm];
  COMPUTE BEFORE Region/STYLE=[JUST=LEFT FONT_WEIGHT=BOLD];
    LINE Region $RegFmt.;
  ENDCOMP;
  BREAK AFTER _Page/PAGE;
  TITLE "Summary of the SASHELP.Shoes dataset";
RUN;
ODS RTF CLOSE;
```

In passing, note that the spanning row headers are created not by the **DEFINE** statement for the **Region** variable, but by the associated **COMPUTE** block. Also note the use of the **INDENT** style attribute, which is new in SAS 9, to obtain an indentation without having to resort to raw RTF code.

5.7 Master-detail presentations: two tables on one page

Source file:	master_detail.sas
Output file:	master_detail.rtf
SAS log file:	master_detail.log
SRS Paginator log file:	master_detail_SRSPaginator.log
SRS Formatter log file:	master_detail_SRSFormat.log

This example will use the case of adverse event data collected during a clinical trial to illustrate how The SRS Pagination Suite can easily be used to create attractive master-detail, or one-to-many, presentations.

In this example, simple demographic information will be presented in a small header table immediately above the main listing table containing details of each adverse event experienced by the subject. The number of adverse events experienced by each subject is unknown, and may flow over more than one page. When more than one page is required to present the information for an individual patient, the ehader table should be repeated at the top of each page, and a continuation footnote should appear on each page save the last. In addition, the space required to present each adverse event may also vary.

Begin by calling the **%paginateCLI** macro to claculate the space required to render each adverse event in the table. In this example, it is obvious that the space required by each observation will be defined by the space required to print the **AEText** variable. Therefore, for simplicity, this is the only variable that is passed to the Paginator.

```
%paginateCLI(indata=DATA.MasterDetail,
             outdata=MasterDetail,
             outdir=&srs_sasdir,
             logfile=&srs_sasdir.master_detail_SRSPaginator.log,
             varlist=  AEText,
             colwidths=7cm,
             tableht=9cm)
```


All material copyright © 2006 Insight Statistical Consulting Ltd

The value of the **tableht** parameter has been chosen to allow for the space required to print the header table and the table titles and footnotes on a page in landscape orientation.

Currently, the header table and the main body table must be created separately. (The advent of a usable tagset-based RTF style in version 9.3 may change this.) Therefore, separate the header data from the that needed for the main body.

```
DATA _Detail (KEEP=_Page PID AEText FromDate ToDate Severity
                   Related Causality Serious Ongoing)
     _Master (KEEP=_Page PID Drug BDate Sex Age Stroke);
  SET MasterDetail END=E;
  BY _Page;
  IF FIRST._Page THEN OUTPUT _Master;
  OUTPUT _Detail;
  IF E THEN CALL SYMPUT("NPAGES", COMPRESS(PUT(_Page, BEST.)));
RUN;
```

Separate calls to the **REPORT** procedure will be used to create the master and detail tables. This means that the data contained in both the **_Master** and **_Detail** tables will need to be subsetted for presentation. This is most easily done in a simple macro. Moreover, the SRSFormatter assumes that when a page with titles but no footnotes is followed by a page with footnotes but no titles, then this is a signal that the tables on the two pages have a master-detail relationship and the page break between the two pages should be removed. Therefore the **TITLE**s and **FOOTNOTE**s will need to be reset before each **REPORT** procedure call.

Start to define the macro. Set the page orientation, close the **LISTING** destination and open the **RTF** destination.

```
%macro do_it;
  OPTIONS ORIENTATION=LANDSCAPE;
  ODS LISTING CLOSE;
  ODS RTF FILE="&srs_sasdir.master_detail.rtf"
          STYLE=styles.LandscapeA4;
```

For each page in the listing…

```
  %do i=1 %to &npages;
```

…work out if the continuation footnote is required.

```
    %let cont=0;
    %if &i lt &npages %then %do;
      PROC SQL NOPRINT;
        SELECT PID INTO :P1 FROM _Master WHERE _Page EQ &i;
        SELECT PID INTO :P2 FROM _Master WHERE _Page EQ %eval(&i+1);
      QUIT;
      %let cont=%eval(&p1 eq &p2);
    %end;
```

Prepare for, and then create, the master table.

```
    TITLE1 "Listing of adverse events";
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
    FOOTNOTE1;
    PROC REPORT DATA=_Master (WHERE=(_Page EQ &i));
      COLUMN PID Drug BDate Sex Age Stroke;
      DEFINE PID/DISPLAY;
      DEFINE Drug/DISPLAY;
      DEFINE BDate/DISPLAY;
      DEFINE Sex/DISPLAY;
      DEFINE Age/DISPLAY;
      DEFINE Stroke/DISPLAY;
    RUN;
```

Prepare for the detail table.

```
    TITLE1;
    %if &cont %then %do;
      FOOTNOTE1 J=L ITALIC "Patient continues overleaf...";
      FOOTNOTE2 J=R "Page &i of &npages";
    %end;
    %else %do;
      FOOTNOTE1 J=R "Page &i of &npages";
    %end;
```

And print the detail table.

```
    PROC REPORT DATA=_Detail (WHERE=(_Page EQ &i));
      COLUMN AEText FromDate ToDate Severity Related Causality
             Serious Ongoing;
      DEFINE AEText/DISPLAY STYLE=[CELLWIDTH=7cm];
      DEFINE FromDate/DISPLAY;
      DEFINE ToDate/DISPLAY;
      DEFINE Severity/DISPLAY LEFT;
      DEFINE Related/DISPLAY LEFT;
      DEFINE Causality/DISPLAY LEFT;
      DEFINE Serious/DISPLAY LEFT;
      DEFINE Ongoing/DISPLAY LEFT;
    RUN;
```

Close the **RTF** destination and re-open the **LISTING** destination.

```
  ODS RTF CLOSE;
  ODS LISTING;
```

Call the SRSFormatter to reformat the file.

```
  X "SRSFormat.exe -in ""&srs_sasdir.master_detail.rtf""
                   -log ""&srs_sasdir.master_detail_SRSFormat.log""";
```

Reset the page orientation and end the macro definition.

```
  OPTIONS ORIENTATION=PORTRAIT;
%mend;
```

Finally, call the macro.

```
%do_it
```

Examination of the master_detail.rtf file will reveal that it is in the required format.

All material copyright © 2006 Insight Statistical Consulting Ltd

The information in the SRSFormatter log file shows the Formatter detected the master-detail relationships correctly:

```
Processing section 1...
  Sections 1 and 2 form a master/detail pair.
Processing section 2...
  Sections 2 and 3 form a master/detail pair.
```

5.8 Using the –sepcols command line parameter

Source file: sepcol_demo.sas
Output file: sepcol_demo.rtf
SAS log file: sepcol_demo.log
SRS Paginator log file: Not applicable
SRS Formatter log file: sepcol_demo_SRSFormatter.log

The **-sepcols** command line parameter allows you to introduce vertical separators between some of the columns in a table. This is not possible using native ODS features. This example shows how to use the SRSFormatter to do this for you.

The table summarises the severity of adverse events experienced by patients in a clinical trial. The procoessing of the data prior to producing the table is omitted: the **AESeverity** dataset holds the contents of the table as it will appear. The **REPORT** procedure call is not directly relvant to the use of the **-sepcols** command line parameter, but it does contain a few useful **REPORT** procedure tricks, which will be explained at the end of this section.

After producing the output RTF file in the usual way, the call to the SRSFormatter includes the parameters **-sepcols (1,7)**, indicating that a single vertical line should appear on the right-hand edges of the first and seventh columns from the left-hand edge of the table. These lines should extend through both the table body and header. The first column is labelled “Body system/preferred term” and the seventh is the one labelled “Sev” and appears in the “Control treatment” spanning group.

The SRSFormatter produces a table which looks, in part, like this:

Severity of adverse events

	Treatment group											
	Control treatment						Active treatment					
Body system/Preferred term	N	%	Mis	Mil	Mod	Sev	N	%	Mis	Mil	Mod	Sev
Subjects treated	59						118					
Body as a Whole	12	20.3	12	2	0	0	96	81.4	63	72	49	10
Abdominal pain	0	0.0	0	0	0	0	10	8.5	2	8	2	0
Abdominal syndrome acute	0	0.0	0	0	0	0	1	0.8	0	0	0	1

Figure 5: An extract from sepcol_demo.rtf, illustrating the effect of the -sepcols command line parameter

All material copyright © 2006 Insight Statistical Consulting Ltd

The –sepcols command line parameter has created the desired visual effect. However, note that the second vertical separator stops at the top of the bottom line of the table header. This is because the first two lines do not have a seventh column as a result of the spanning headers that they contain.

It is worth noting that the use of the **-sepcols**, **-bsepcols** or **-hsepcols** command line parameters slows the performance of the SRSFormatter considerably. To demonstrate this, the **-fulltimer** command line option was added to the call to the SRSPaginator in **sepcol_demo.sas**. The following extract from the corresponding log file shows that the use of –sepcol has increased the time taken to process page 1 of the output file by a factor of about five.

```
Processing section 2...
    Time taken to add column separators: 25.922s.
    Time taken to reformat table:        30.343s.
  Time to process section: 30.531s.
```

There are a couple of interesting features of the REPORT procedure call that produces the table shown above. The first is the use of the <style-attribute>=<SAS format> in some of the column definitions. For example:

```
DEFINE Pct1/DISPLAY "%" FORMAT=F6.1 STYLE=[FOREGROUND=ForeFmt.];
```

This syntax says that, for example, the value of the **FOREGROUND** style attribute is not fixed, but instead is defined by applying the **ForeFmt.** format to the value of **Pct1** in each observation. The **ForeFmt.** format (defined in **define formats.sas**) has the following definition:

```
VALUE ForeFmt .="WHITE" OTHER="BLACK";
```

In other words, the foreground colour should be black if a non-missing value is found and white otherwise. Thus, the full stops usually used to represent missing numeric values can be suppressed.

An extension of this technique can be found in the **COMPUTE** block associated with the **AEP** variable. In this case, the values of a different variable is used to define a style attribute for the **AEP** variable. In this case, the need is to print body system headers in bold and preferred terms within body system indented by 0.5cm. The input dataset is organised so that the value of **AEP** is set to the value of **BodySystem** for the first observation within each body system group. The following code snippet achieves the desired effect.

```
COMPUTE AEP;
  IF AEBody EQ AEP THEN
    CALL DEFINE(_COL_, "STYLE", "STYLE=[FONT_WEIGHT=BOLD]");
  ELSE
    CALL DEFINE(_COL_, "STYLE", "STYLE=[INDENT=0.5cm]");
ENDCOMP;
```

6 The Client/Server Utility

All material copyright © 2006 Insight Statistical Consulting Ltd

The SRS Pagination Suite includes a basic client/server implementation to allow the suite to be run remotely. This makes the reformatting facilities of the SRS Formatter avialable on any node of a network on which at least one Windows based node is available, regardless of the operating system that an individual node is running.

6.1 Installing the client/server utility

The server executables (**`SRSServer.exe`** and **`SRSServer.sh`**) is located in *The SRS Pagination Suite'*s installation directory. No further installation steps are necessary.

The client executables (**`SRSClient.exe`** and **`SRSClient.sh`**) is also placed in *The SRS Pagination Suite'*s installation directory. To install the client on a client host, simply copy the appropriate executable, together with the **`client.jar`** JAR file, to a folder on the client host.

To install the client on a host running an operating system other than Microsoft Windows, Unix or Linux, copy the **`client.jar`** JAR file to the client host and write a shell file that runs it.

6.2 Running the server process

The SRS Pagination Suite server is invoked with the following command:

`<srsserver> <command line paramaters>`

where

`<srsserver>`	is **`SRSServer.exe`** on Windows, **`SRSServer.sh`** on Unix and Linux, or the appropriate user-supplied shell file on other operating systems.
`<command line parameters>`	are as described below.

The command line parameters recognised by The SRS Pagination Suite server are defined in Section 6.3 below.

6.3 Parameters of the server process

6.3.1 -port <port number>

Required. Defines the port on which the server should listen for incoming server requests. Default **`4444`**.

6.3.2 -debug <n>

Optional. Sets the debug level, which controls the amount of information written to the server log file. Default **`3`**.

The table below summarises the information written to the log for various values of **`-debug`**:

All material copyright © 2006 Insight Statistical Consulting Ltd

Debug level	Information provided
0	Initiation, termination, error and warning messages Client connection and disconnection Client connection validation failure
1	Addition and removal of a job to the job queue
2	Not used
3	Notification of starting a job The command used to execute a job Notification of the successful completion of a job
4	Before execution of a job, list the command line parameters as received by the server
5	Monitoring of parameter requests as they are received from the client

Table 6: Debug settings for the SRS Pagination Suite server

6.3.3 -log <log file name>

Optional. Specifies the name of the server log file. The default name is **`SRSServer_<hostname>_<username>_<date/time>.log`**

where

`<hostname>` is the name of the host on which the server process is running,
`<username>` is the name of the user who set the server process running and
`<date/time>` is the date and time at which the server process started, in the format **`yyyymmdd_hhmmss`**.

6.3.4 -srsformat <file name>

Optional. Defines the path to the SRSFormat executable. Default **`SRSFormat.exe`**. Since the SRS formatter installer adds the installation directory to the Windows path variable, this should be sufficient to locate the file.

6.3.5 -srspaginate <file name>

Optional. Defines the path to the SRSPaginator executable. On Windows systems, the default is **`SRSPaginateCLI.exe`**. Otherwise, the default is **`SRSPaginate.sh`**.

> *Note to Windows users: as the SRS Paginator installation routine is designed to be generic across operating systems, it does not modify the windows path. Therefore, the default value of* **`-srspaginate`** will not be sufficient *to locate the executable file unless either (a) you have subsequently modified the value of the* **`PATH`** *environment variable to include the SRS Paginator's installation directory or (b) you installed the paginator in a folder that was already included in the* **`PATH`** *environment variable.*

**** Question to beta testers: what is the situation on Unix/Linux?

6.3.6 –echo

All material copyright © 2006 Insight Statistical Consulting Ltd

Requests that output produced by SRSFormat or SRSPaginator jobs is echoed to the server log as well as to the log file for the individual job. The default is **–noecho**.

See also: **-noecho**.

6.3.7 –noecho

Requests that output produced by SRSFormat or SRSPaginator jobs is not echoed to the server log as well as to the log file for the individual job. This is the default.

See also: **-echo**.

6.4 The SRS server log file

Entries in the server log file are time stampled and, when not gnerated by the main server thread, include a prefix that indcates the thread that generated them. The information contained in the log is best explained by a series of examples.

6.4.1 Example 1: A single conncetion request

In this first example, the server is started, a single job is received from a client and the server is then terminated.

The first four lines are standard. They indicate the name of the log file, the name of the host and the port on which server is listening for connections.

```
13Apr2006 17:37:04 **** NOTE: Log file is 'D:\Program Files
\Insight\SRS Pagination
Suite\SRSPaginator\SRSServer_Insight2_John_20060413_173704.log'.
13Apr2006 17:37:04 **** NOTE: SRS Server v1.0 started at 13Apr2006
17:37:04.
13Apr2006 17:37:05 **** NOTE: Listening for SRS Client connections on
Insight2, port 4444.
13Apr2006 17:37:05 **** NOTE: Hit <ENTER> to terminate.
```

The next two lines indicate that a connection attempt has been received and a server thread has been started to service the request. The second line indicates that the client has successfully transmitted its request and dropped the connection. The time between the two messages – less than a second – is typical as the client requests are extremely siimple.

```
13Apr2006 17:37:23 [Server thread 0] **** NOTE: Connection from
INSIGHT1 [192.168.0.3] accepted.
13Apr2006 17:37:24 **** NOTE: Connection closed at client request.
```

Next, the sever thread adds the job request to its job queue. This triggers the job queue monitor thread, which is currently idle, to request the server thread to pass the job to the job queue monoitor thread. The second line indicates that the server thread has done so.

```
13Apr2006 17:37:24 Added job 'Bert' to the queue.  There is now 1 job
pending.
```


All material copyright © 2006 Insight Statistical Consulting Ltd

```
13Apr2006 17:37:24 Removed job 'Bert' from the queue.  There are now
0 jobs pending.
```

The next message are generated by the job queue monitor and indicate that an SRS Formatter process has successfully been created.

```
13Apr2006 17:37:24 [Job queue monitor] Starting formatter job 'Bert'.
```

When the SRS Formatter job completes, the job queue monitor reports this fact and then shuts down.

```
13Apr2006 17:37:57 [Job queue monitor] Job 'Bert' completed without
error.  Size of job queue is now: 0
```

Finally, the sever acknowledges the user's shut down request.

```
13Apr2006 17:38:21 **** NOTE: Termination request received.
```

6.4.2 Example 2: A second connection request arrives before the first job has completed

The log file starts in exactly the same way as the previous example.

```
13Apr2006 17:53:54 **** NOTE: Log file is 'D:\Program Files
\Insight\SRS Pagination
Suite\SRSPaginator\SRSServer_Insight2_John__20060413_175353.log'.
13Apr2006 17:53:54 **** NOTE: SRS Server v1.0 started at 13Apr2006
17:53:54.
13Apr2006 17:53:54 **** NOTE: Listening for SRS Client connections on
Insight2, port 4444.
13Apr2006 17:53:54 **** NOTE: Hit <ENTER> to terminate.

13Apr2006 17:54:03 [Server thread 0] **** NOTE: Connection from
INSIGHT1 [192.168.0.3] accepted.
13Apr2006 17:54:04 **** NOTE: Connection closed at client request.

13Apr2006 17:54:04 Added job 'Bert' to the queue.  There is now 1 job
pending.
13Apr2006 17:54:04 Removed job 'Bert' from the queue.  There are now
0 jobs pending.
13Apr2006 17:54:04 [Job queue monitor] Starting formatter job 'Bert'.
```

At this point, a second client request is received and processed by a new server thread. (The server thread index only takes account of the threads that are running *concurrently*.)

```
13Apr2006 17:54:18 [Server thread 0] **** NOTE: Connection from
INSIGHT1 [192.168.0.3] accepted.
13Apr2006 17:54:18 **** NOTE: Connection closed at client request.

13Apr2006 17:54:18 Added job 'Eric' to the queue.  There is now 1 job
pending.
```

A short while later, the first job completes successfully.

All material copyright © 2006 Insight Statistical Consulting Ltd

```
13Apr2006 17:54:33 [Job queue monitor] Job 'Bert' completed without
error.  Size of job queue is now: 1
```

This allows the job monitor queue to start processing the second job request.

```
13Apr2006 17:54:33 Removed job 'Eric' from the queue.  There are now
0 jobs pending.
13Apr2006 17:54:33 [Job queue monitor] Starting formatter job 'Eric'.
```

Again, this completes without error.

```
13Apr2006 17:55:01 [Job queue monitor] Job 'Eric' completed without
error.  Size of job queue is now: 0
```

And finally the server is terminated by the user.

```
13Apr2006 17:55:08 **** NOTE: Termination request received.
```

6.4.3 Example 3: Two client requests arrive almost simultaneously

**** To be completed.

6.5 Running a client process

An SRS client process is executed by a SAS job in a similar way to running the SRS Paginator or SRS Formatter locally. That is, by using a SAS X command with the following format:

```
<srsclient> <command line paramaters>
```

where

`<srsclient>`	is **`SRSClient.exe`** on Windows, **`SRSClient.sh`** on Unix and Linux, or the appropriate user-supplied shell file on other operating systems.
`<command line parameters>`	are as described in section 6.6 below.

6.6 Parameters of the SRS client utility

The command line parameters recognised by The SRS Pagination Suite client are described in the follwing subsections. Any other pramaters found on the command line are passed to the SRS server and are used as parameters to the job that the server is requested to run.

> *Note: when file names are passed to the server, they must be specified using paths that are valid for the server's environment. For example, if a user on host* **`Insight1`** *requests that* **`test.rtf`** *on local drive* **`F:`** *is to be reformatted, the correct syntax is*

```
-srsformatter –in \\Insight1\F\test.rtf
```


All material copyright © 2006 Insight Statistical Consulting Ltd

6.6.1 -server <servername>

Required. The name of the server on which the SRS server is currently running.

6.6.2 -port <port number>

Required. Default: **4444**. The port on which the SRS server is currently listening.

6.6.3 -debug <debug level>

Optional. Default: **3**. The debug level with which the client should run.

Debug level	Information provided
0	Error and warning messages.
1	Not used
2	Not used
3	Shows the acknowledgement of the request from the server.
4	Not used
5	Logs each command line parameter as it is sent to the server.

6.6.4 -clientlog <client log file name>

Optional. The name of the file to which the client log is to be written. The default log file name is

SRSClient_<username>_<date/time>.log

where

<username> is the name of the user invoking the client,

<date/time> is the date and time at which the client was invoked, in the format **yyyymmdd_hhmmss**.

6.6.5 –srsformat

Specifies that the client is requesting the server to run the SRS Formatter.

6.6.6 -srspaginate

Specifies that the client is requesting the server to run the SRS Paginator.

6.6.7 -bye

Required. Signals to the SRS server that no more parameters are forthcoming. Thus, this ***must*** be the final paramter on the command linepassed to the server. This parameter is supplied by the client automatically.

6.6.8 -srsformatter

All material copyright © 2006 Insight Statistical Consulting Ltd

Optional. Signifies that the client is requesting an SRS Formatter job. Exactly one of **-srsformatter** and **-srspaginator** must be present.

6.6.9 -srspaginator

Optional. Signifies that the client is requesting an SRS Paginator job. Exactly one of **-srsformatter** and **-srspaginator** must be present.

6.7 The client log file

The SRS Client utility provides an extremely simple log file. Here is an example of a log file showing a sucessful connection request:

```
[13Apr2006 16:32:45] **** NOTE: Connected to Insight2 on port 4444.
[13Apr2006 16:32:51] Server responded with 'Thank you'.
[13Apr2006 16:32:51] **** NOTE: Disconnected from Insight2.
[13Apr2006 16:32:51] **** NOTE: Done.
```

The second line shows the server's acknowledgement of a properly formatted request.

If the server is not running at the time the client requests a connection, a log file similar to the following is created:

```
[13Apr2006 17:24:00] **** FATAL ERROR: Unable to get I/O stream for
server 'Insight2', port 4444.
```

6.8 The SRS client/server utility: examples

**** To be completed ****

6.9 Licencing the client/server utility

The SRS Pagination Suite's client/server utility is freeware. No fee is payable to licence it. However, you must still agree to abide by the terms of *The SRS Pagination Suite*'s End User Licence Agreement (EULA) before using any part of *The SRS Pagination Suite*.

The SRS Pagination Suite itself is licenced software. You will need a separate licence for each server on which it is installed.

7 Trouble shooting

Error 1904. Module C:\WINNT\system32\mstcf.dll failed to register. HRESULT –2147024769. Contact your support personnel.

All material copyright © 2006 Insight Statistical Consulting Ltd