The statrep package[*]

Tim Arnold and Warren F. Kuhfeld

SAS Institute Inc.

`tim.arnold@sas.com`

`warren.kuhfeld@sas.com`

2015/09/15

Contents

---

[*]This document corresponds to statrep v1.07, last revised 2015/09/15.

List of Figures

1   About This Document

This document uses the following aids for presenting information.

- The verbatim font (`example text`) is used for names of variables, macros, and environments.

- Hyperlinks are displayed in red.

- Figures are displayed with a light blue background.

- Macro names are given in the margin near where the macro is defined or described.

- The index contains terms with pointers to the code lines.

2   StatRep Usage

To use the StatRep package, add the `\usepackage` command to your document preamble after you declare the `documentclass`.

Figure 1 shows an example of using the StatRep package and specifying two of its three options.

```
\documentclass{book}
\usepackage[figname=output,resetby=chapter]{statrep}
```

Figure 1: Example of Using the StatRep Package

The StatRep package supports the following options:

- `generate` specifies whether a SAS program is generated at compile time. It can have a value of `true` or `false`; the default is `true`.

- `figname=` specifies the name of a LaTeX counter that is used for numbering outputs. The default is `figure`. If you specify a value for the `figname` option for which no counter exists, a counter is created.

- `resetby=` specifies that the counter for output numbering be reset with each change in the specified counter value. For example, if `resetby=chapter`, all output numbering is reset when the chapter value changes. See section 4 for details. Also, refer to the `tocloft` package documentation for information about how the lists are typeset.

The options `figname=` and `resetby=` are not used directly by the StatRep package but are passed to the `longfigure` package.

The `longfigure` package is provided with the StatRep package. It supports display and page breaking within a stream of outputs, and it can be used independently of the StatRep package. It supports the options `figname=` and `resetby=`. For complete details about the implementation of the `longfigure` package, see section 4.

In Figure 1, the specified options label the included output as appropriately numbered `Outputs` and enable a *List of Outputs* to be generated. To generate a *List of Outputs*, add the `\listofoutput` command at the point in your document where you want the *List of Outputs* to appear. If you specify `figname=display`, add the `\listofdisplay` command where you want the *List of Displays* to appear.

If you load the StatRep package with no options, the outputs are labeled as figures and you can display the *List of Figures* with the command `\listoffigures`.

For more information about how to use the StatRep package, see the StatRep User's Guide (`statrepmanual.pdf`) that accompanies the StatRep distribution.

3 StatRep Implementation

3.1 Required Packages

The StatRep package requires the following LaTeX packages. Each package is available in a relatively recent TeXLive distribution (2005 or later).

```
1 \RequirePackage{verbatim}
2 \RequirePackage{graphicx}
3 \RequirePackage{xkeyval}
4 \RequirePackage{calc}
5 \RequirePackage{ifthen}
```

- The verbatim package provides the foundation for the StatRep package.

- The graphicx package enables inclusion of images and is used by the \Graphic tag.

- The xkeyval, calc, and ifthen packages provide programming capabilities that are used throughout the StatRep package.

```
 6 \newif\ifSR@generate\SR@generatetrue
 7 \DeclareOptionX{generate}[true]{\@nameuse{SR@generate#1}}
 8 \DeclareOptionX{color}[true]{%
 9   \IfFileExists{sas.sty}
10   {\PassOptionsToPackage{\CurrentOption}{sas}}
11   {\relax}
12 }
13 \DeclareOptionX{figname}{\PassOptionsToPackage{\CurrentOption}{longfigure}}
14 \DeclareOptionX{resetby}{\PassOptionsToPackage{\CurrentOption}{longfigure}}
15 \ProcessOptionsX
16 \IfFileExists{sas.sty}
17   {\RequirePackage{sas}}
18   {\relax}
19 \RequirePackage{longfigure}
```

Declare an option for color; Pass the value to the sas package if it exists, otherwise do nothing. This option has no effect if the ODS Listing destination is used. It applies only to SAS-generated LaTeX tabular output.

The file sas.sty is loaded if it exists. The file is created by a SAS ODS tagset that is experimental; the tagset generates SAS output as LaTeX tables. More information on this experimental support can be found in the User's Guide (appendix).

4

3.2 Programming Utilities

The programming helpers described in this section are used throughout the package:

```
20 \ifSR@generate\def\SR@writepgm{\immediate\write}
21 \else\let\SR@writepgm\@gobbletwo
22 \fi
```

\SR@generate \SR@generate is a Boolean switch that is created and defined to have a default value of true. The switch is used to provide alternative definitions of a
\SR@writepgm utility macro \SR@writepgm. If the switch remains true, \SR@writepgm is defined as \immediate\write; otherwise, it is defined to remove its two arguments (\@gobbletwo).

The effect of these alternative definitions is as follows. When you do not override the default (by default, generate is true), the macro writes to a file stream that represents the generated program file. Otherwise, the macro is defined to remove its two arguments (therefore, it is effectively a null operation).

The following two commands set the tolerance for bad boxes, orphans, and widows:

```
23 \newcommand*{\dosloppy}{\setlength{\hfuzz}{\maxdimen}\hbadness\maxdimen}
24 \newcommand*{\unsloppy}{\setlength{\hfuzz}{0.3pt}\hbadness 1414}
```

\dosloppy The dosloppy macro is intended to be used when you knowingly violate type-
\unsloppy setting rules, such as when you must write extremely wide outputs. For such situations, set the tolerance high with dosloppy and set it back to normal with unsloppy. These settings do not change LATEX's line-breaking algorithms or line penalties; they suppress the overfull box (\hfuzz) and underfull box (\hbadness) warnings in the log. The system uses theses commands when it insert outputs. See the \SR@set@outmargin macro in section 3.5.6 for details.

The following definitions are used to calculate and set the output stream of listings or figures. See section 3.5.6 for details.

```
25 \newlength{\SR@scratchlength}
26 \newlength{\SR@verbwidth}
27 \newsavebox{\SR@filebox}
28 \newcommand*{\SR@firsthead}{}
29 \newcommand*{\SR@conthead}{}
30 \newcommand*{\SR@endfoot}{}
```

The following variables aid in constructing the two verbatim environments provided in the StatRep package:

```
31 \newcounter{SR@currentline}\setcounter{SR@currentline}{0}
32 \newcounter{SR@displayline}\setcounter{SR@displayline}{0}
33 \newcounter{SR@programline}\setcounter{SR@programline}{0}
34 \newcounter{SR@startinglastline}\setcounter{SR@startinglastline}{0}
35 \newcounter{SR@totallines}\setcounter{SR@totallines}{0}
36 \newcounter{SR@multifilecount}\setcounter{SR@multifilecount}{0}
37 \newboolean{SR@multifile}\setboolean{SR@multifile}{false}
```

- SR@currentline contains the current line number of the verbatim environment that is being processed.

- SR@displayline contains the number of lines to display. The number can be specified as a line command inside a Sascode environment.

- SR@programline contains the number of lines to write to the generated program. The number can be specified as a line command inside a Sascode environment.

- SR@startinglastline contains the line number that begins the bottom block of an abbreviated displayed Datastep environment. This counter is used when the last= option is specified in the Datastep environment.

- SR@totallines contains the total number of lines in the verbatim environment.

- SR@multifilecount counter is used to step through and insert a set of output files that are generated by SAS.

- SR@multifile is a Boolean switch that is used in the algorithm that inserts the stream of outputs. See section 3.5.6 for complete details.

```
38 \begingroup\catcode`\#=12\gdef\SR@hashchar{#}\endgroup
```

\SR@hashchar The \SR@hashchar macro is used in writing options that are specified in the \Listing and \Graphic macros. Options that specify SAS ODS object names can contain a hash character. In the LaTeX source, the normal hash character (#) must be escaped (\#). When the string is written to the generated program, the escaping backslash must be removed. The definition resets the category code inside a group so that the change will not affect other definitions. Then the \SR@hashchar macro is globally defined as an unescaped hash character.

The \SR@hashchar macro is used in the \SR@write@outoptions macro. See section 3.5 for details.

SR@ keys   The following SR@ family of keys is used in the `Datastep` and `Sascode` environments and the `\Listing` and `\Graphic` macros.

```
39 \define@boolkeys{SR}[SR@]{display,program,continued}[true]{}
40 \define@cmdkeys{SR}[SR@]{caption,first,last}{}
41 \define@cmdkeys{SR}[SR@]{fontsize,scale,label}{}
```

The keys are used for two purposes: One set of keys is used by the StatRep package to typeset the environments or outputs that are defined in the package. Another set of keys is used to communicate with SAS; the keys are passed through directly to the generated program. There is some overlap between the two set of keys. For example, the `width` key is used by the StatRep package to typeset the output that is specified in a `\Graphic` tag and is used by SAS to generate the image.

The following keys are used only by the StatRep package:

- `display` is a Boolean key that specifies that a `Sascode` or `Datastep` block should only be displayed.

- `program` is a Boolean key that specifies that a `Sascode` or `Datastep` block should only be written to the program file.

- `continued` is a Boolean key that specifies whether the output is a continuation of a preceding output block.

- `caption=` specifies the caption to use for a `Listing` or `Graphic` output.

- `first=` specifies the number of top lines to display in a `Datastep` environment.

- `last=` specifies the number of bottom lines to display in a `Datastep` environment.

- `fontsize=` specifies the LaTeX font size to use in displaying output or code environment (for example, `small` or `footnotesize`).

- `scale=` specifies a factor by which to scale a `Graphic` image. For example, specify `scale=0.5` to scale the image to half its original size, or specify `scale=2` to scale it to double its original size.

- `label=` is used internally by the StatRep package to generate a label for `\Graphic` and `\Listing` elements.

The following keys are used both by the StatRep package and by SAS:

- `store=` specifies the name of the ODS document store to be created in a `Sascode` environment. When this key is specified, the StatRep package writes the appropriate SAS program lines to the generated program. When this key is not specified, the StatRep package assumes that the author has written the SAS macro calls into the `Sascode` blocks.

- `linesize=` specifies the line size that is used to generate and typeset `Listing` output. Typical values are 80, 96, or 120.

- `width=` specifies the width to generate or display `Graphic` output.

- `dest=` specifies the SAS ODS destination to use for tabular output. The possible values are `listing` and `latex`. The `listing` option includes verbatim text-based tables from the ODS LISTING destination. The `latex` option includes LaTeX tabular environments that use the `sas` package (generated during the SAS run).

```
42 \define@cmdkeys{SR}[SR@]{store,linesize,width,dest}{}
```

The remaining keys are defined and accepted by the `\Listing` and `\Graphic` macros, but are passed on to the generated SAS program. The following keys are used only by SAS:

```
43 \define@cmdkeys{SR}[SR@]{type,style,dpi}{}
44 \define@cmdkeys{SR}[SR@]{firstobj,lastobj}{}
45 \define@cmdkeys{SR}[SR@]{objects,pattern,options}{}
46 \define@cmdkeys{SR}[SR@]{height,pagesize}{}
```

- `style=` specifies the ODS style to use in generating output.

- `dpi=` specifies how many dots per inch (DPI) to use in generating a graph.

- `firstobj=` specifies the first object's name to capture in an output stream.

- `lastobj=` specifies the last object's name to capture in an output stream.

- `objects=` specifies a space-separated list of object names to capture.

- `pattern=` specifies a name-matching pattern to select objects to capture.

- `options=` specifies other options for generating output.

- `height=` specifies the height of a generated graph (for example, `height=5.4in`).

- `pagesize=` specifies page size for output.

- `type=` specifies the type of output stream to initiate (`graphic` or `listing`). If the automatic program generation capability is in effect (that is, the `store=` option is specified), the `\Listing` and `\Graphic` tags automatically generate the appropriate value for this key.

`\presetkeys` When any environment or command uses the keys, it first calls the following `presetkeys` macro, which provides default values for each key:

```
47 \presetkeys{SR}{%
48    display  = false,   program  = false,   continued = false,
49    caption  = \@empty, first    = 0,        last      = 0,
50    fontsize = \@empty, scale    = \@empty, label     = \@empty,
51    store    = \@empty, linesize = \@empty, width     = \@empty,
52    type     = \@empty, style    = \@empty, dpi       = \@empty,
53    dest     = \@empty, firstobj = \@empty, lastobj   = \@empty,
54    objects  = \@empty, pattern  = \@empty, options   = \@empty,
55    height   = \@empty, pagesize = \@empty}{}
```

3.3 Customizable Settings

The definitions in this section can be overridden by settings in the external file `statrep.cfg`.

`\SRmacropath` `\SRmacropath` specifies the path to the location of the SAS macros that are bundled with the StatRep package. You can define the path by using forward slashes (/) instead of backslashes (\) as the directory name delimiter. If you do use backslashes, you must insert a backslash character into this argument by using the `\@backslashchar` command. The default value is the current directory. That is, the definition for the path to the macro file is the filename itself, `statrep_macros.sas`.

```
56 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
57 %% Edit this line to point to the location of the StatRep macros.
58 \def\SRmacropath{statrep_macros.sas}
59 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The following statements define line size, page size, graphic resolution, and also specify the default ODS output style:

`\SRlinesize`
`\SRpagesize`
`\SRdpi`
`\SRstyle`
`\SRodsgraphopts`
`\SRlatexstyle`

```
60 \def\SRrootdir{.}
61 \def\SRlinesize{80}
62 \def\SRpagesize{500}
63 \def\SRdpi{300}
```

9

```
64 \def\SRstyle{statistical}
65 \def\SRlatexstyle{statrep}
66 \def\SRodsgraphopts{}
```

The following statements define system options used in the generated SAS program

- `\SRlinesize` specifies the default linesize to use for listing output.

- `\SRpagesize` specifies the default pagesize to use for listing output.

- `\SRdpi` specifies the default DPI setting to use for GRSEG and ODS graphics.

- `\SRstyle` specifies the default ODS output style to use for ODS graphics.

- `\SRlatexstyle` specifies the ODS output style to use for LaTeX tabular output. The default is `statrep`, which is provided in the supplmental file `statrep_tagset.sas`.

```
67 \def\SRgraphicdir{\SRrootdir/png}
68 \def\SRgraphtype{png}
69 \def\SRlistingdir{\SRrootdir/lst}
70 \def\SRlatexdir{\SRrootdir/tex}
71 \def\SRdefaultdests{listing}
```

`\SRgraphicdir`
`\SRgraphtype`
`\SRlistingdir`
`\SRdefaultdests`
`\SRlatexdir`

- `\SRgraphicdir` specifies the name of the directory that contains the graphic output files that are generated by SAS. The default is `png`.

- `\SRgraphtype` specifies the type of graphics file that SAS will generate. The choices are `png` and `pdf`. The default is `png`.

- `\SRlistingdir` specifies the name of the directory that contains the listing (tabular) output files that are generated by SAS. The default is `lst`.

- `\SRlatexdir` specifies the name of the directory that contains the LaTeX (tabular) output files that are generated by SAS. The default is `tex`.

- `\SRdefaultdests` specifies the ODS destination to use for tabular output. The possible values are `listing` and `latex`. The default is `listing`.

```
72 \def\SRprogramname{\jobname_SR.sas}
73 \def\SRpreamblename{\jobname_SR_preamble.sas}
74 \def\SRmacroinclude{\@percentchar include "\SRmacropath" /nosource;}
75 \def\SRsasprogramline{%
```

```
76 \@percentchar cleandir(\SRrootdir, tex, tex);^^J%
77 \@percentchar cleandir(\SRrootdir, png, png);^^J%
78 \@percentchar cleandir(\SRrootdir, lst, lst);^^J%
79 }%
```

\SRprogramname
\SRmacroinclude
\SRprogramline

- \SRprogramname specifies the filename for the generated SAS program. The default is \jobname_SR.sas, where \jobname is usually the stem name of the LaTeX source file.

- \SRmacroinclude specifies the line used in the generated SAS program to include the SAS macros that are bundled with the StatRep package. The default is %include \SRmacropath /nosource;.

- \SRprogramline specifies the first lines to include in the generated SAS program after the \SRmacroinclude line. The following default value calls a SAS macro that removes the contents of the listing, tex, and graphic directories. The directories are created with each SAS run that includes the macros themselves (via x commands).

```
80 \def\SRparindent{3em}
81 \def\SRintertext{... more data lines ...}
82 \def\SRtempfilename{sr.tmp}
83 \def\SRcontinuedname{continued}
84 \def\SRcaptionfont{\sffamily}
85 \def\SRcaptioncontinuedfont{\sffamily\itshape}
86 \def\SRverbfont{\ttfamily\bfseries}
```

\SRparindent
\SRintertext
\tempfilename

- \SRparindent specifies the amount of space to indent Datastep and Sascode environments. The argument is a dimension. The default is 3em and is measured according to the font currently in use.

- \SRintertext specifies the text to insert in abbreviated Datastep environments (that is, Datastep environments that specify the first= option).

- \SRtempfilename specifies the name of a temporary file that is used as a scratch file in the current working directory. The default is sr.tmp.

\SRcontinuedname
\SRcaptionfont
\SRcaptioncontinuedfont
\SRverbfont

- \SRcontinuedname specifies the name that indicates that an output block is continued. This helper is used when an output stream breaks across a page. The default is continued.

11

- \SRcaptionfont specifies the font to use for the output captions. The default is \sffamily (sans serif).

- \SRcaptioncontinuedfont specifies the font to use for the continued name for outputs that break across pages. The default is \sffamily\itshape (*sans serif, italic*).

- \SRverbfont specifies the font to use for code within Datastep and Sascode blocks. The default is \ttfamily\bfseries (typewriter text, bold).

If the file statrep.cfg exists, the following statements load it so that you can override the preceding definitions:

```
87 \InputIfFileExists{statrep.cfg}
88   {\PackageInfo{statrep}{Reading custom configuration file statrep.cfg}}
89   {\PackageInfo{statrep}{No custom configuration file found (statrep.cfg).%
90                          Using defaults.}}
```

Two output streams are created:

```
91 \newwrite\SR@program@stream
92 \newwrite\SR@tempfile@stream
```

The stream \SR@program@stream represents the generated SAS program. The stream \SR@tempfile@stream represents a temporary stream and is used to write the SAS program preamble and to construct each Datastep environment.

The following lines define boilerplate text that is included in the preamble SAS file and the capture-program SAS file:

```
 93 \def\SR@preambletext{/*^^J
 94   This file is auto-generated by the statrep package.^^J
 95   Do not edit this file or your changes will be lost.^^J
 96   Edit the LaTeX file instead.^^J
 97   ^^J
 98   See the statrep package documentation and the file^^J
 99   statrep.cfg for information on these settings.^^J
100   */^^J
101   ^^J
102 }
```

The following statements open the temporary stream by using the filename \SR@preamblename and write the default settings into the preamble file. The temporary file is then closed.

```
103 \def\SR@write@preamble{
104    \immediate\openout\SR@tempfile@stream\SRpreamblename%
105    \SR@writepgm\SR@tempfile@stream{\SR@preambletext}
106    \SR@writepgm\SR@tempfile@stream{/* Set and invoke macro variable defaults. */}
107    \SR@writepgm\SR@tempfile@stream{\@percentchar let rootdir=\SRrootdir;}
108    \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultstyle=\SRstyle;}
109    \SR@writepgm\SR@tempfile@stream{\@percentchar let latexstyle=\SRlatexstyle;}
110    \SR@writepgm\SR@tempfile@stream{\@percentchar let graphtype=\SRgraphtype;}
111    \SR@writepgm\SR@tempfile@stream{\@percentchar let graphicdir=\SRgraphicdir;}
112    \SR@writepgm\SR@tempfile@stream{\@percentchar let listingdir=\SRlistingdir;}
113    \SR@writepgm\SR@tempfile@stream{\@percentchar let latexdir=\SRlatexdir;}
114    \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultlinesize=\SRlinesize;}
115    \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultpagesize=\SRpagesize;}
116    \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultdpi=\SRdpi;}
117    \SR@writepgm\SR@tempfile@stream{\@percentchar let odsgraphopts=\SRodsgraphopts;}
118    \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultdests=\SRdefaultdests;}
119    \SR@writepgm\SR@tempfile@stream{}
120    \SR@writepgm\SR@tempfile@stream{options nodate nonumber}
121    \SR@writepgm\SR@tempfile@stream{ls=&defaultlinesize ps=&defaultpagesize}
122    \SR@writepgm\SR@tempfile@stream{formchar='|----|+|---+=|-/\@backslashchar<>*';}
123    \SR@writepgm\SR@tempfile@stream{}
124    \SR@writepgm\SR@tempfile@stream{ods graphics on;}
125    \SR@writepgm\SR@tempfile@stream{/* Include SAS macro definitions. */}
126    \SR@writepgm\SR@tempfile@stream{\SRmacroinclude}
127    \SR@writepgm\SR@tempfile@stream{}
128    \immediate\closeout\SR@tempfile@stream
```

The following statements begin writing the generated SAS program \SRprogramname.
They open the \SR@program@stream, write lines to include the preamble file,
and do the preliminary work necessary before each run of the capture program.

The preamble is written to a separate file and not directly into the capture program
so that users who copy and paste code from the LaTeX source to a SAS session can
include only the preamble in their SAS session.

```
129    \immediate\openout\SR@program@stream\SRprogramname
130    \SR@writepgm\SR@program@stream{\SR@preambletext}
131    \SR@writepgm\SR@program@stream{\@percentchar include "\SRrootdir/\SRpreamblename" /no
132    \SR@writepgm\SR@program@stream{/* Remove all output files. */}
133    \SR@writepgm\SR@program@stream{\SRsasprogramline}
134    \SR@writepgm\SR@program@stream{}
135    \SR@writepgm\SR@program@stream{/* Start program with a null title. */}
136    \SR@writepgm\SR@program@stream{title;}
137    \SR@writepgm\SR@program@stream{}
138 }
```

\SR@write@preamble The `\SR@write@preamble` macro opens the `\SR@tempfile@stream` stream and writes the preamble for the SAS program to be generated. The preamble is constructed from the preceding configurable macros (for example, `\SRlinesize`, `\SRdpi`). You can modify the preamble in the `statrep.cfg` file.

```
139 \AtBeginDocument{\SR@write@preamble}
140 \AtEndDocument{\immediate\closeout\SR@program@stream}
```

At the beginning of the document, the `\SR@program@stream` stream is opened and the program file preamble (that is, the initial definitions described previously) is written to the program file. At the end of the document, the `\SR@programout` stream is closed.

3.4   Code Environments

The StatRep package depends on the `verbatim` package to handle its code environments. The `verbatim` package provides programming hooks for redefining commands that are used to process `verbatim` environments. By redefining some of the commands that are called by the `verbatim` package, the StatRep package can preprocess and postprocess code blocks as needed.

The `verbatim` package defines the following macros as hooks for processing `verbatim` environments:

- `\verbatim@addtoline` adds its argument to the character buffer.

- `\verbatim@processline` typesets the characters that accumulate in the buffer.

By redefining these macros, the StatRep package can control the reading, processing, and writing of the `verbatim` environment. The `Datastep` environment redefines `\verbatim@processline` at the beginning and the end of the code block. The `Sascode` environment redefines `\verbatim@addtoline` by inserting several line-processing steps before adding characters to the character buffer.

3.4.1   Datastep Environment

Three macros are defined to set up the `Datastep` environment:

```
141 \def\SR@datastep@writefile#1{%
142   \setcounter{SR@totallines}{0}%
143   \@bsphack\immediate\openout\SR@tempfile@stream#1%
```

14

```
144   \let\do\@makeother\dospecials\catcode'\^^M\active%
145   %
146   \def\verbatim@processline{%
147     \addtocounter{SR@totallines}{1}%
148     \SR@writepgm\SR@tempfile@stream{\the\verbatim@line}%
149   }%
150   \verbatim@start%
151 }
```

\SR@datastep@writefile   The SR@datastep@writefile macro is used at the beginning of the Datastep
environment. It resets category codes, opens a temporary file, and redefines
\verbatim@processline to count the lines in the environment and write the
code block to a temporary file.

The logic in the \SR@datastep@writefile macro is taken from an example in
the verbatim package documentation (Schöpf, Raichle, and Rowley 2001).

```
152 \def\SR@enddatastep@writefile{\immediate\closeout\SR@tempfile@stream\@esphack}
```

\SR@enddatastep@writefile   The \SR@enddatastep@writefile macro closes the temporary file.

```
153 \newcommand{\SR@datastep@writeprogramline}{%
154   \SR@writepgm\SR@program@stream{\the\verbatim@line}}
```

\SR@datastep@writeprogramline

The \SR@datastep@writeprogramline macro writes a line from the code
block to the generated program. It is used when the code block is not spec-
ified as display-only. The Datastep environment begins by redefining the
\verbatim@processline macro to write the code block to a temporary file. It
also keeps a count of the total number of lines in the block. It ends by again re-
defining the \verbatim@processline macro, this time to process lines from the
temporary file according to options that are specified in the Datastep environ-
ment.

Figure 2 shows how a Datastep environment is processed. The names in blue
represent macros that are called by the verbatim package, even though the macro
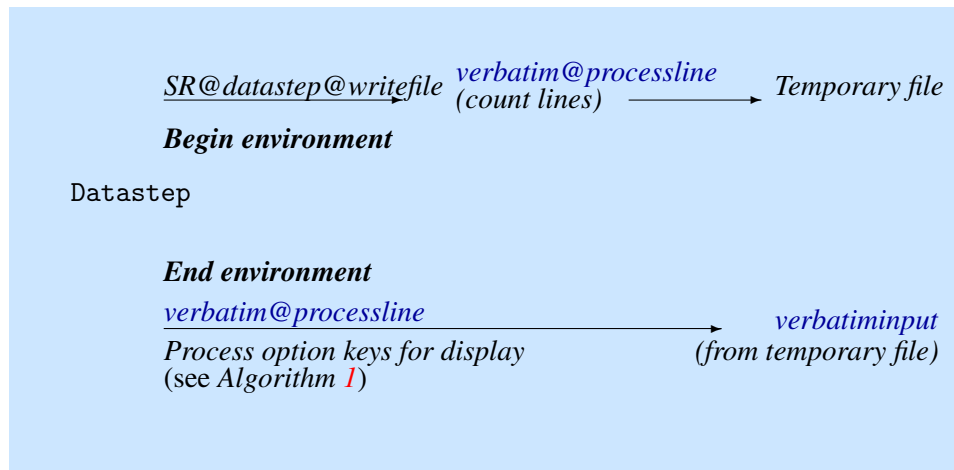contents might have been redefined.

SR@datastep@writefile   *verbatim@processline*   → *Temporary file*
                  (count lines)

Begin environment

Datastep

End environment
verbatim@processline
Process option keys for display   → *verbatiminput*
(see *Algorithm 1*)   *(from temporary file)*

Figure 2: Processing Schematic for the `Datastep` Environment

```
155 \def\@Datastep[#1]{%
156   \setkeys{SR}{#1}%
157   \SR@datastep@writefile{\SRtempfilename}%
158 }
159 \newenvironment{Datastep}{%
160   \catcode'\^^M=\active
161   \@ifnextchar[{\catcode'\^^M=5\@Datastep}{\catcode'\^^M=5\@Datastep[]}
162   }
```

Datastep The beginning of the `Datastep` environment sets the key-value options that are specified for the environment and calls the StatRep macro `\SR@datastep@writefile`.

If you specify a `Datastep` environment with no options, and you place a comment character at the beginning of the first line of the environment, that line is hidden from the display. LATEX looks for the first "real" character of the environment and does not see the commented line. To account for this possiblity, the `\@Datastep` command is used internally to begin the real environment. At the initialization of the `Datastep` environment, the category code for the end-of-line character is changed to *active*. If the environment has options, processing continues as normal (`\@Datastep` is called directly). If the environment has no options, a blank set of brackets is inserted into the stream and the `\@Datastep` is then called. Thus, if you specify a `Datastep` environment with no options, LATEX knows not to continue looking for the end of the command because the empty brackets represent an empty set of options.

```
163 {%
```

```
164    \SR@enddatastep@writefile\endgraf%
```

The end of the `Datastep` environment performs the following tasks:

1. calls `\SR@enddatastep@writefile` to close the temporary file

2. redefines `\verbatim@processline` to handle line processing

3. includes the temporary file that uses those redefined macros

If the block should only be written to the program and not displayed, redefine `\verbatim@processline` to `\SR@datastep@writeprogramline` (defined previously) to write a line to the generated program file. Then process the temporary file.

```
165    \ifSR@program\def\verbatim@processline{\SR@datastep@writeprogramline}%
166    \else%
```

Otherwise, the code block is to be displayed.

```
167       \ifthenelse{\equal{\SR@fontsize}{\@empty}}%
168       {\relax}%
169       {\@nameuse{\SR@fontsize}}%
170       \setcounter{SR@currentline}{0}%
171       \setcounter{SR@startinglastline}{\theSR@totallines-\SR@last-1}%
```

Set the font size to be used in the display. Set the `SR@currentline` counter to 0. Set `SR@startinglastline` to the total number of lines in the environment minus the number of bottom lines to display.

`\verbatim@processline` Redefine the `\verbatim@processline` macro to account for options that are specified to the environment and read in the temporary file.

```
172       \def\verbatim@processline{%
```

If the code block is not display-only, write each line to the program file by using the `\SR@datastep@writeprogramline` macro, defined previously.

```
173          \ifSR@display\else\SR@datastep@writeprogramline\fi%
```

The remainder of the macro logic is described in Algorithm 1 on page 19.

Supporting the `last=` option in the `Datastep` environment is the only reason for the complexity of writing to a temporary file and retrieving the lines from the file. To know how to display the last n lines, StatRep must know the total number of

lines in the code block. So it writes the code block to the temporary file and keeps a tally of the number of lines as it writes each line.

A horizontal skip of length \SRparindent is added to the beginning of each \the\verbatimline line that is displayed.

A \par is also appended to each \the\verbatimline: In a verbatim environment, the entire block is typeset as a list of one item, and within that item, each line is a paragraph. The following statements append a \par to each \the\verbatimline to end the paragraph:

```
174        \ifnum\SR@first>0%
175          \ifnum\theSR@currentline<\SR@first%
176            {\leavevmode\hskip\SRparindent\the\verbatim@line\par}%
177          \else%
178            \ifnum\theSR@currentline=\SR@first%
179              {\leavevmode\par\hskip\SRparindent\SRintertext\par}%
180            \else%
181              \ifnum\SR@last>0%
182                \ifnum\theSR@currentline=\theSR@startinglastline{\leavevmode\par}\fi%
183                \ifnum\theSR@currentline>\theSR@startinglastline%
184                  {\leavevmode\hskip\SRparindent\the\verbatim@line\par}%
185                \fi%
186              \fi%
187            \fi%
188          \fi%
189        \else%
190          {\leavevmode\hskip\SRparindent\the\verbatim@line\par}%
191        \fi%
192        \addtocounter{SR@currentline}{1}%
193      }%
194    \fi%
```

The following algorithm describes how this redefined version of the \verbatim@processline

macro processes each line.

```
 1  for each line in block do
 2      if block is not display only then
 3          write data line ;
 4      if first=n is specified then
 5          if currentline < n then
 6              display line;
 7          else if currentline = n then
 8              write the value of \SRintertext;
 9          else if last=m is specified then
10              if currentline = (totallines - m) then
11                  start new line;
12              else if currentline > (totallines - m) then
13                  display line;
14      else
15          display line;
16  end
```

Algorithm 1: Datastep Environment: \verbatim@processline Macro

When all settings are configured, and when subsidiary macros and the \verbatim@processline macro are redefined, the following statement reads in the temporary file that was created at the beginning of the Datastep environment. The file is read by using the verbatim package macro \verbatiminput. That macro then calls \verbatim@processline, which the StatRep package has just redefined.

```
195    \verbatiminput{\SRtempfilename}%
```

The code block has now been displayed and written to the program file as specified by the options in the Datastep environment. However, there is still cleanup left to do.

The following statements determine whether the block was displayed: if it was not displayed, they retrieve all vertical space that is associated with the environment:

```
196    \ifSR@program
197    \setlength{\SR@scratchlength}{-2\topsep-\partopsep-2\parskip-\baselineskip}
198    \vspace*{\SR@scratchlength}%
199    \fi%
```

If the block is written to the program file, the following statements output a blank

line after this `Datastep` environment in the program file:

```
200  \ifSR@display\else%
201    \SR@writepgm\SR@program@stream{}%
202  \fi%
203 }
```

3.4.2 Sascode Environment

The following macros are defined to set up the `Sascode` environment:

- `\SR@sascode@addtoline` resets category codes and calls `\SR@sascode@filter`.

- `\SR@sascode@filter` acts as a filter. It processes lines with line commands and passes other lines to `\SR@sascode@writeline`.

- `\SR@sascode@writeline` writes the lines into the character buffer that is used by the `verbatim` package. It also writes lines to the generated program file.

The `Sascode` environment uses this sequence of macros by replacing the `verbatim` package macro `\verbatim@addtoline` with the `\SR@sascode@addtoline` macro.

The following statements start a group to safely change the category code for the % character to *other*. Within this group, the % character can be used in other definitions because its category code is no longer the comment category code (normally category code 14).

```
204 \begingroup
205   \catcode`\%12
```

`\SR@sascode@addtoline` The following statements call the `\SR@sascode@addtoline` macro, reset category codes, and call `\SR@sascode@filter` with two arguments: the original category code of the % character and the verbatim line itself. The original category code for the % character is sent so that it can be returned to its original state after `\SR@sascode@filter` is called.

```
206  \gdef\SR@sascode@addtoline{\catcode`\%12
207       \expandafter\relax
208       \expandafter\SR@sascode@filter
209       \expandafter{\number\catcode`\% }}
```

20

After expansion, the following command is left, where *n* is the original category code of the % character:

`\relax\SR@sascode@filter` *n*

That is, the macro first redefines the category code of the % character and calls the `\SR@sascode@filter` macro with the original category code of the % character as its first argument.

\SR@sascode@filter   The `\SR@sascode@filter` macro does the work of filtering the code block for line commands. See Algorithm 2 for details.

This macro is the reason StatRep requires the use of pdfLATEX: the PDF primitive `\pdfmatch` is essential to the macro.

\pdfmatch   The `\pdfmatch{`⟨*pattern*⟩`}{`⟨*string*⟩`}` command implements pattern matching (using the syntax of POSIX regular expressions). The first argument is a regular expression pattern and the second argument is a string. The command expands to –1 if the pattern is invalid, to 0 if no match is found, and to 1 if a match is found. The primitive was introduced in pdfTEX 1.30.0 (Thanh et al. 2009). The result of `\pdfmatch` is stored in an array.

\pdflastmatch   The `\pdflastmatch{`⟨*integer*⟩`}{`⟨*expandable*⟩`}` returns the match that corresponds to `{`⟨*integer*⟩`}` provided by the `\pdfmatch` command. Entry 0 contains the full match, and further entries contain submatches (captured group) that correspond to the subpatterns of the match.

The regular expression argument must be escaped for use in LATEX. For example, consider the task to capture the integer in the following expression:

`%* program 4;`

A typical regular expression to capture the integer 4 is defined as follows:

`^%\*\s*program\s*([0-9]+);`

To use the expression in LATEX, it must be escaped so that the pattern becomes

`^%\string\*\space*program\space*([0-9]+);`

That is, `\*` becomes `\string\*` and `\s*` becomes `\space*`.

The second argument is detokenized so that it represents a string of characters for use in the `\pdfmatch` macro.

The effect of the `\SR@sascode@filter` macro is that every line that contains a line command is parsed, the appropriate line counters are set, and the line is skipped.

Algorithm 2 describes how the `\SR@sascode@filter` processes each line of the

21

Sascode block.

```
1  for each line in block do
2      if line begins with %* (column 1) then
3          if line matches %* program n;
4          then
5              set program counter
6          else if line matches %* display n;
7          then
8              set display counter
9          else if line matches %* ;
10         then
11             write line to program;
12         else
13             display and write to program (valid SAS macro comment)
14         end
15     else
16         pass line onward to \SR@sascode@writeline;
17     end
18 end
```

Algorithm 2: Sascode Environment: SR@sascode@filter Macro

```
210    \gdef\SR@sascode@filter#1#2{
211      \ifnum\pdfmatch{^%\string\*}
212                    {\detokenize{#2}}=1
213        \ifnum\pdfmatch{^%\string\*\space*program\space*([0-9]+);}
214                      {\detokenize{#2}}=1
215          \setcounter{SR@programline}
216                      {\expandafter\strip@prefix\pdflastmatch 1}
217          \catcode`\%#1\@tempswafalse\verbatim@line{}
218        \else
219          \ifnum\pdfmatch{^%\string\*\space*display\space*([0-9]+);}
220                        {\detokenize{#2}}=1
221            \setcounter{SR@displayline}
222                        {\expandafter\strip@prefix\pdflastmatch 1}
223            \catcode`\%#1\@tempswafalse\verbatim@line{}
224          \else
225            \ifnum\pdfmatch{^%\string\*\space*;(.*)$}
226                          {\detokenize{#2}}=1
227              \SR@writepgm\SR@program@stream{\expandafter\strip@prefix\pdflastmatch 1}
228              \catcode`\%#1\@tempswafalse\verbatim@line{}
229            \else \catcode`\%#1\SR@sascode@writeline{#2}
```

```
230            \fi
231          \fi
232        \fi
233      \else
234        \catcode`\%#1\SR@sascode@writeline{#2}
235      \fi
236    }
237 \endgroup
```

The \begingroup that started the definition block for \SR@sascode@addtoline
and \SR@sascode@filter is closed with \endgroup.

In the \SR@sascode@filter macro, the Boolean switch \tempswa must be set
to false for skipped lines because of how the verbatim package uses the switch.
In the \verbatim@processline macro, the switch determines whether to insert
a line ending. That is, it decides whether a new verbatim environment is starting
(\@tempswatrue). This decision determines whether to end a paragraph. Lines
are equivalent to paragraphs inside a verbatim environment.

Because the Sascode environment redefines the \verbatim@addtoline macro
and not the \verbatim@processline macro from the verbatim package, the
Sascode environment must set the \tempswa switch so that it is set appropriately
when the verbatim package invokes \verbatim@processline. In this way, the
\tempswafalse macro ensures that the displayed verbatim environment does not
have blank lines in places where a line command was used.

The \SR@sascode@filter macro passes all lines that are not line commands to
the \SR@sascode@writeline macro. Lines that begin with the special line com-
mand, the null SAS macro comment (%*;), are written directly to the generated
program file after stripping the macro comment.

The final **else** statement is executed when neither the program counter nor the
display counter is set.

\SR@sascode@writeline   The \SR@sascode@writeline macro tests whether line counters are set and
writes each line to its appropriate destination (the display or the generated pro-
gram file or both). See Algorithm 3 for a description of the macro's behavior.

Note the use of \SRparindent to indent the code lines so that the Sascode envi-
ronment display matches the Datastep environment display.

```
238 \newcommand{\SR@sascode@writeline}[1]{%
239   \ifSR@program%
240     \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
```

```
241      \@tempswafalse%
242    \else%
243      \ifSR@display%
244        \verbatim@line\expandafter{\the\verbatim@line\hskip\SRparindent#1}%
245      \else%
246        \ifnum\theSR@programline>0%
247          \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
248          \addtocounter{SR@programline}{-1}%
249          \@tempswafalse%
250        \else%
251          \ifnum\theSR@displayline>0%
252            \verbatim@line\expandafter{\the\verbatim@line\hskip\SRparindent#1}%
253            \addtocounter{SR@displayline}{-1}%
254          \else%
255            \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
256            \verbatim@line\expandafter{\the\verbatim@line\hskip\SRparindent#1}%
257          \fi%
258        \fi%
259      \fi%
260    \fi%
261 }
```

Algorithm 3 describes how the \SR@sascode@writeline macro processes each
line that is received from the \SR@sascode@filter macro.

1	**for** *each line received* **do**		
2		**if** *program only* **then**	
3			write line to program file
4		**else if** *display only* **then**	
5			display line
6		**else if** *Program counter set* **then**	
7			write line to program file;
8			decrement \SR@program counter;
9		**else if** *Display counter set* **then**	
10			display line;
11			decrement \SR@display counter;
12		**else**	
13			write line to program file;
14			display line;
15		**end**	
16	**end**		

Algorithm 3: Sascode Environment: SR@sascode@writeline Macro

Figure 3 shows how a `Sascode` environment is processed. The names in blue represent macros that are called by the `verbatim` package, even though the macro contents might have been redefined.
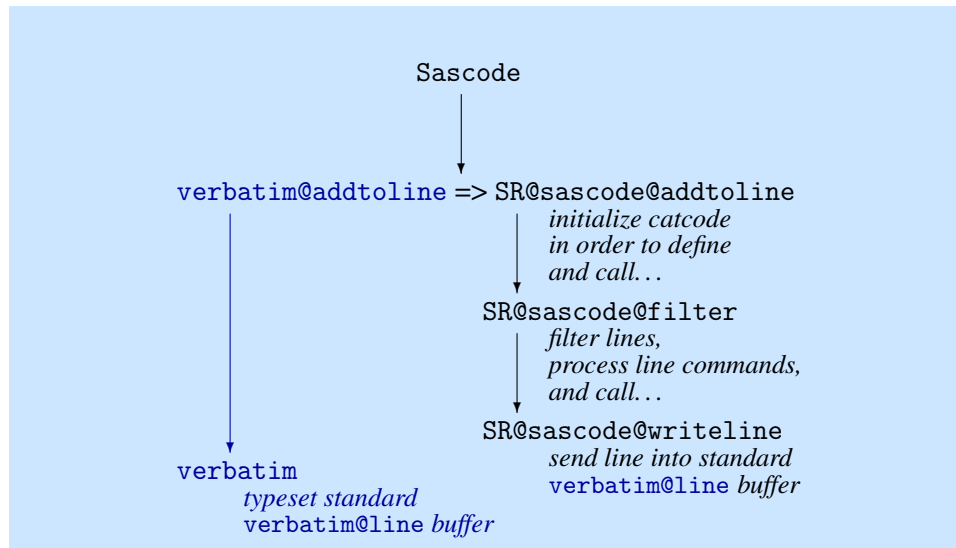
Figure 3: Processing Schematic for the `Sascode` Environment

Sascode The `Sascode` environment first sets the values for the optional keys. If the environment is to be written to the generated program, the appropriate keys are processed and passed to the program.

If you do not use the `store=` key, StatRep assumes that you are writing your own SAS macros to output the ODS document and write the outputs. If you do use the key, the macros are written automatically.

The central point of the code for processing the `Sascode` environment is to redirect the `verbatim` package macro, `\verbatim@addtoline`, to the StatRep package macro,`\SR@sascode@addtoline`. This redirection begins the workflow shown in Figure 3. The processing steps are as follows:

1. The `verbatim` package macro calls `\verbatim@addtoline`.

2. The `\verbatim@addtoline` macro redirects to `\SR@sascode@addtoline`.

3. The `\SR@sascode@addtoline` macro sets category codes and calls `\SR@sascode@filter`.

4. The `\SR@sascode@filter` macro processes line commands and calls `\SR@sascode@writeline`.

5. The `\SR@sascode@writeline` macro parses and processes the lines. It writes the appropriate lines to the generated program file. It also writes the appropriate lines into the `\verbatim@line` buffer, which is processed by the verbatim package macro `\verbatim`.

If you specify a Sascode environment with no options, and you place a comment character at the beginning of the first line, that line is hidden from the display. LATEX looks for the first "real" character of the environment and does not see the commented line. To account for this possiblity, the `\@Sascode` command is used internally to begin the real environment. At the initialization of the Sascode environment, the category code for the end-of-line character is changed to *active*. If the environment has options, processing continues as normal (`\@Sascode` is called directly). If the environment has no options, an empty set of brackets is inserted into the stream and `\@Sascode` is then called. Thus, if you specify a Sascode environment with no options, LATEX knows not to continue looking for the end of the command because the empty brackets represent an empty set of options.

```
262 \def\@Sascode[#1]{%
263   \setkeys{SR}{#1}%
264   \ifSR@display\else%
265     \ifthenelse{\equal{\SR@store}{\@empty}}%
266       {\relax}%
267       {\SR@writepgm\SR@program@stream{\@percentchar output(\SR@store)}%
268        \PackageInfo{statrep}{Processing Sascode block \SR@store}}%
269   \fi%
270   \ifthenelse{\equal{\SR@fontsize}{\@empty}}%
271     {\relax}%
272     {\@nameuse{\SR@fontsize}}%
273   \setcounter{SR@programline}{0}%
274   \setcounter{SR@displayline}{0}%
275   \let\verbatim@addtoline\SR@sascode@addtoline%
276   \verbatim}
277 \newenvironment{Sascode}{%
278   \catcode`\^^M=\active
279   \@ifnextchar[{\catcode`\^^M=5\@Sascode}{\catcode`\^^M=5\@Sascode[]}
280 }
```

The end of the Sascode environment performs the last steps that are needed and does some cleanup. The following statements remove vertical space that is caused by the end of the Sascode environment and write the %endoutput macro contents to the generated program file:

```
281 {\endverbatim%
282   \setlength{\SR@scratchlength}{-2\topsep-\partopsep-2\parskip}
283   \vspace*{\SR@scratchlength}%
284   \ifSR@program\vspace*{-\baselineskip}\fi
285   %
286   \ifSR@display\else%
287     \ifthenelse{\equal{\SR@store}{\@empty}}%
288       {\relax}%
289       {\SR@writepgm\SR@program@stream{\@percentchar endoutput(\SR@store)}%
290        \SR@writepgm\SR@program@stream{}}%
291   \fi%
292 }
```

3.5   Handling the SAS Output

The \Listing and \Graphic tags support options that are used in the generated
program and in output display.

There are three possible cases for an output request:

- The output is of type graphic.

- The output is of type listing and destination listing.

- The output is of type listing and destination latex.

When the store= option is used in the tags, a line is written to the %write macro
that passes a subset of the options in the tag to the generated program. The tags
also insert the stream of specified outputs in the displayed document.

Graphical output is inserted as specified in the \Graphic tag, captioned, and cen-
tered.

Listing output is inserted in the same way, but the output is framed when the
destination is listing; when the destination is latex, the output (which is in the
form of LaTeX tabular environments) is not framed.

The frame for the listing destination is created with the help of the longfigure
package and the \hline command. If the ODS destination is LaTeX, the resulting
output blocks are put into a minipage of \textwidth width and are not framed.

The following helper macros are used only in writing to the generated program:

\SR@write@outoptions is called when the store= option is specified in the
    output tag. The command parses the options that are specified in the output

tag for program-related options and writes the appropriate SAS macro to the generated program.

\SR@set@outoptions creates a string of key=value pairs that are used in the generated SAS macro.

\SR@final@outoptions accumulates each key=value pair into one string.

\SR@outkeyval tests each key to determine whether to add the key and its value to the final option string.

\SR@write@outoptions If the store= option is specified in a \Listing or \Graphic tag, the \SR@write@outoptions macro is called. Otherwise, nothing is written to the generated SAS program. For each line that is written, use the macro \SR@hashchar to write out an unescaped hash character (#) when an escaped hash character (\#) is read. See section 3.2 for information about the \SR@hashchar macro.

```
293 \def\SR@write@outoptions{
294     \SR@set@outoptions
295     \let\#\SR@hashchar
296     \SR@writepgm\SR@program@stream{%
297         \@percentchar write(\SR@label,store=\SR@store \SR@final@outoptions)
298     }
```

After the program line is written, a blank line is output to the program:

```
299     \SR@writepgm\SR@program@stream{}
300 }
```

\SR@set@outoptions The following statements define the \SR@set@outoptions macro. This macro resets the \SR@final@outoptions definition and calls \SR@outkeyval for the keys that can be passed through to the generated SAS program. This processing is necessary so that keys with \@empty values are not passed to the generated program.

```
301 \newcommand*{\SR@set@outoptions}{%
302     \global\edef\SR@final@outoptions{}
303     \SR@outkeyval{height}\SR@outkeyval{objects}\SR@outkeyval{pattern}
304     \SR@outkeyval{style}\SR@outkeyval{dpi}\SR@outkeyval{options}
305     \SR@outkeyval{pagesize}\SR@outkeyval{linesize}\SR@outkeyval{width}
306     \SR@outkeyval{dest}\SR@outkeyval{type}\SR@outkeyval{firstobj}\SR@outkeyval{lastobj}
307     }
```

Recall that all keys are set to \@empty with the \presetkeys macro. If the value has not been changed from the default, the following statements omit it from the

\SR@final@outoptions macro definition.

\SR@outkeyval The \SR@outkeyval macro expands \SR@final@outoptions to a string of key=value pairs. With each call to the \SR@outkeyval macro, the argument is checked to determine whether it is empty. If it is not empty, the argument is expanded and added to the \SR@final@outoptions value.

```
308 \newcommand*{\SR@outkeyval}[1]{%
309   \def\SR@outkey{#1}
310   \edef\SR@outval{\csname SR@#1\endcsname}
311   \ifx\@empty\SR@outval
312   \else
313     \edef\SR@final@outoptions{\SR@final@outoptions,\SR@outkey=\SR@outval}
314   \fi
315   }
```

The following macros are used to display the SAS generated output.

\Listing inserts a stream of listing outputs.

\Boxlisting boxes a single listing.

\Graphic inserts a stream of graphical outputs.

\Boxgraphic boxes a single graphic.

\SR@insert begins the output insertion process.

\SR@set@outheadings sets the heading for a longfigure.

\SR@set@outmargin sets the left margin for centering a longfigure.

3.5.1 Note on Widths

If the output is a listing and the destination is listing, the output will be inserted as verbatim type input. The width is calculated by the linesize and the output centered by adjusting the \LFleft margin. The width calculation is done once, at the \Listing command itself.

If the output is a graphic, the output is put into a box at whatever scale or width the user specified. The width of that box is calculated and the output centered by adjusting the \LFleft margin. The width calculation cannot be done at the \Graphic command since the output must be measured first. It is in the \Boxgraphic macro that the calculation takes place, which means it happens for

each graphic in the output stream. Only the first graphic has an effect; it is only the first graphic in the stream that affects the `\LFleft` margin. This is an inefficiency.

If the output is a listing and the destination is `latex`, the output cannot be measured at all. The output is put into a minipage of `textwidth` width and the `\LFleft` margin is untouched. The user can therefore set `\LFleft` before the `\Listing` tag to shift the output if needed.

3.5.2 The `\Listing` **Macro**

`\Listing` The `\Listing` macro sets the options that are specified in the tag and automatically creates a label by using its {⟨*argument*⟩}, which serves also as the name of the output file (without an extension).

```
316 \newcommand{\Listing}[2][]{%
317   \setkeys{SR}{#1,type=listing,label=#2}%
318   \ifthenelse{\equal{\SR@store}{\@empty}}%
319     {\relax}%
320     {\SR@write@outoptions}%
```

The `type=` option is automatically specified as `listing`; this option is passed to the generated SAS program. If the `store=` option is specified, the `\SR@write@outoptions` macro is invoked to write the appropriate SAS macro to the generated file.

```
321 \ifthenelse{\equal{\SR@dest}{\@empty}}%
322     {\def\SR@dest{\SRdefaultdests}}%
323     {\relax}%
```

If no value was given for the `dest=` option, set `\SR@dest` to the default value (specified globally as `\SRdest`). Now we are guaranteed to have a value for `\SR@dest` (used in the `\Boxlisting` and `\SR@insert` macros).

The next block only pertains to the situation where we have a verbatim listing output from ODS Listing destination. We need to find the width of the output, which is done by calculating the width of a single character in the specified font and multiplying that value by the number of characters per line (linesize). This line length is saved in the value `\SR@scratchlength` and used to center the output.

```
324 \ifthenelse{\equal{\SR@dest}{listing}}%
325 %    The following statements set the font size:
326 %      \begin{macrocode}
327   {\ifthenelse{\equal{\SR@fontsize}{\@empty}}%
328     {\def\SR@fontsize{normalsize}}%
```

```
329      {\relax}%
```

The following statements set \SR@scratchlength to the width of a single character (~) by using the current font size and the verbatim font:

```
330      \settowidth{\SR@scratchlength}{\@nameuse\SR@fontsize%
331                                    \SRverbfont\selectfont~}%
```

The following statements control the line size for ODS Listing output:

```
332    \ifthenelse{\equal{\SR@linesize}{\@empty}}%
333      {\def\SR@linesize{\SRlinesize}}{\relax}%
```

The value for the key \SR@linesize is set to \@empty with each invocation of the \Listing command. If no value is specified for \SR@linesize, the following statements set it to the specified default value for line size \SRlinesize. This default is set to 80 by the StatRep package; you can set the line size in the statrep.cfg file.

The linesize= key corresponds to the line size that the SAS program uses to generate the tabular output. By default it is set to 80; other typical values are 96 and 120.

The following statements set \SR@verbwidth to the width of \SR@linesize number of columns, with each column as wide as \SR@scratchlength. The result represents the width of the contents of a SAS listing output.

```
334      \setlength{\SR@verbwidth}{(\SR@linesize\SR@scratchlength)}%
```

Finally, the following statements add 2\tabcolsep and 2\arrayrulewidth to that width. The result represents the total width of a \Listing output and is used in the \SR@insert macro to center the output.[1]

```
335      \setlength{\SR@scratchlength}{\SR@verbwidth+2\tabcolsep+
336                                              2\arrayrulewidth}%
337    }% end if dest is listing
338    {\relax}% do nothing if dest is latex.
```

With \SR@verbwidth calculated, \SR@insert is called to insert the \Listing. When the Listing is being set, the value for the left margin of the longfigure is calculated. After the Listing is finished, reset the value back to its default, 0.

---

[1]A listing output stream gets its width once, at the beginning of the stream. A graphic gets the width for each graphic file in its stream. Only the first measurement is used to calculate the graphic width (that is, only the first graphic file width is used).

```
339    \SR@insert{#2}%
340    \setlength{\LFleft}{0in}%
341 }
```

3.5.3   The \Graphic **Macro**

The \Graphic macro sets the options that are specified in the tag and automatically creates a label by using its {⟨*argument*⟩}, which also serves as the name of the output file (without an extension).

```
342 \newcommand{\Graphic}[2][]{%
343    \setkeys{SR}{#1,type=graphic, label=#2}%
344    \ifthenelse{\equal{\SR@store}{\@empty}}%
345      {\relax}%
346      {\SR@write@outoptions}%
347    \SR@insert{#2}%
348    \setlength{\LFleft}{0in}%
349 }
```

\Graphic   The type= option is automatically specified as graphic; this option is passed to the generated SAS program. If the store= option is specified, the \SR@write@outoptions macro is invoked to write the appropriate SAS macro to the generated file.

Finally, \SR@insert is called to insert the \Graphic. When the Graphic is being set, the value for the left margin of the longfigure is calculated. After the Graphic is finished, reset the value back to its default, 0.

3.5.4   The \Boxlisting **Macro**

If the \SRdest value has been set for this listing, the value for \SR@dest now contains the value (see the Listing code above).

If the current destination is listing, the \Boxlisting macro creates a box that contains the verbatim content of an external listing file. Put the content of the specified file into a box as verbatim input. If the file doesn't exist, place a marker in the document.

If the current destination is latex, the \Boxlisting macro puts the contents of the specified file into a minipage and then into the box.

```
350 \newcommand*{\Boxlisting}[1]{%
351    \ifthenelse{\equal{\SR@dest}{listing}}%
```

32

```
352        {\global\def\SR@insertname{lst/#1.lst}%
353         \IfFileExists{\SR@insertname}%
354           {%
355              \savebox{\SR@filebox}{%
356                 \def\verbatim@font{%
357                     \@nameuse{\SR@fontsize}%
358                     \SRverbfont%
359                     \hyphenchar\font\m@ne%
360                     \@noligs%
361                     }%
362                 \parbox{\SR@verbwidth}{%
363                   \vspace*{0.25\baselineskip}%
364                   \noindent\verbatiminput{\SR@insertname}%
365                   \vspace*{0.25\baselineskip}}%
366              }%
367           }%
368           {\savebox{\SR@filebox}{\fbox{Missing File \SR@insertname}}%
369            \global\def\SR@insertname{\@empty}}%
370        }%
371        {\global\def\SR@insertname{tex/#1.tex}%
372         \IfFileExists{\SR@insertname}%
373          {%
374             \savebox{\SR@filebox}{%
375                \begin{minipage}{\textwidth}%
376                   \vspace{0.25\baselineskip}%
377                   \centering\input{\SR@insertname}%
378                   \vspace{0.25\baselineskip}%
379                \end{minipage}%
380             }%
381          }%
382          {\savebox{\SR@filebox}{\centering\fbox{Missing File \SR@insertname}}%
383           \global\def\SR@insertname{\@empty}%
384          }%
385        }%
386     \global\setbox\SR@filebox=\box\SR@filebox%
387     \PackageInfo{statrep}{Processing output \SR@insertname}%
388  }
```

\Boxlisting The \Boxlisting macro takes a single argument, the {⟨*filename*⟩} of the list-
ing output to be boxed (the filename without the extension). If the file exists,
the macro places the content of the file (using \verbatiminput) into a box of
\SR@verbwidth, which is calculated as described in section 3.5.2. White space
at the top and bottom of the box is inserted (0.25\baselineskip).

The `\verbatim@font` is set to the user-defined `\SR@fontsize` (from the options on the Listing tag) and the `\SRverbfont` in the configuration. Furthermore, ligatures and the hyphenation character are both turned off. If you omit this step, ligatures will appear in the listing output (the case which prompted this bug-fix was a dashed line; the emdash ligature was applied to each pair of dashes).

Note that `\SR@verbwidth` is the width that is used to create the `\parbox` that contains the output. `\SR@scratchlength` is used to center the output in a later step. `\SR@scratchlength` represents the width of the output plus the surrounding frame.

If the file does not exist, `\SR@insertname` is defined to be `\@empty` and `\SR@filebox` is defined to be a box with the contents `Missing File`.

3.5.5 The `\Boxgraphic` **Macro**

The `\Boxgraphic` macro creates a box that contains the content of an external graphic file.

```
389 \newcommand*{\Boxgraphic}[1]{%
390   \IfFileExists{png/#1.\SRgraphtype}%
391     {\global\def\SR@insertname{png/#1.\SRgraphtype}}%
392     {\global\def\SR@insertname{\@empty}}%
393   \ifthenelse{\equal{\SR@insertname}{\@empty}}%
394     {\savebox{\SR@filebox}{\fbox{ Missing File png/#1.\SRgraphtype}}}%
395     {%
396       \ifthenelse{\equal{\SR@scale}{\@empty}}%
397         {\ifthenelse{\equal{\SR@width}{\@empty}}%
398           {\savebox{\SR@filebox}{\includegraphics{\SR@insertname}}}%
399           {\savebox{\SR@filebox}{\includegraphics[width=\SR@width]%
400                                                   {\SR@insertname}}}}%
401         {\savebox{\SR@filebox}{\includegraphics[scale=\SR@scale]%
402                                                 {\SR@insertname}}}%
403     }%
404   \global\setbox\SR@filebox=\box\SR@filebox%
405   \PackageInfo{statrep}{Processing graphic \SR@insertname}%
406   \setlength{\SR@scratchlength}{\wd\SR@filebox+2\tabcolsep}%
407 }%
```

`\Boxgraphic`   The `\Boxgraphic` macro takes a single argument, the {⟨*filename*⟩} of the graphic output to be boxed (the filename without the extension).

The macro first checks whether the file exists. If it does, it sets the global variable

`\SR@insertname` to the full name. If the file does not exist, `\SR@insertname` is defined to be `\@empty`.

If the global variable `\SR@insertname` is empty, then `\SR@filebox` is defined to be a framed box with the contents `Missing File`.

Otherwise, we use the `\includegraphics` macro from the `graphicx` package to place the image.

If the `width=` or `scale=` option was specified, the option is passed to the `\includegraphics` macro.

Finally, `\SR@scratchlength` is set to the width of the graphic (now contained in `\SR@filebox`) + 2\tabcolsep. The result represents the total width of a `\Graphic` output and is used in the `\SR@insert` macro to center the output.

3.5.6 The `\SR@insert` Macro

The `\SR@insert` macro sets headings and footers for a `longfigure`, begins the `longfigure` environment, and calls the appropriate macro to display an output stream by using `\Boxlisting` or `\Boxgraphic`. The `longfigure` package is described in detail in section 4.

SR@filebox Two helper macros are necessary to set up for the `\SR@insert` command:

`\SR@set@outheadings` sets the headings on the output.

`\SR@set@outmargin` sets the left margin to center the output.

\SR@set@outheadings The `\SR@set@outheadings` macro sets the contents of the `longfigure`'s first heading, its "continued" heading, and the footer. If the content is a `\Listing` and the destination is `listing`, the `longfigure` is framed (the headings and footers include an `\hline` command).

```
408 \def\SR@set@outheadings{%
```

The following statements define three macros for typing shortcuts:

```
409    \def\SR@@caption{\caption{\SRcaptionfont\SR@caption}}%
410    \def\SR@@captioncontinued{\caption{\SRcaptioncontinuedfont\SRcontinuedname}}%
411    \def\SR@@captionskip{\\*[0.5\baselineskip]}%
```

The following statements define a set of \SR@@ macros that are used as calculators for the actual macros used at the time of typesetting:

```
412    \ifthenelse{\equal{\SR@dest}{listing}}%
413              {\def\SR@@firsthead{\hline\endLFfirsthead}%
414               \def\SR@@conthead{\hline\endLFhead}%
415               \def\SR@@endfoot{\hline\endLFfoot}}%
416              %
417              {\def\SR@@firsthead{\endLFfirsthead}%
418               \def\SR@@conthead{\endLFhead}%
419               \def\SR@@endfoot{\endLFfoot}}%
420    %%%
```

If the \SR@dest is listing, \hline commands are used for the heading and footer of the longfigure.

The following statements set initial values for the heading and footer based on the preceding definitions:

```
421    %%%
422    \renewcommand*{\SR@firsthead}{\SR@@firsthead}%
423    \renewcommand*{\SR@conthead}{\SR@@conthead}%
424    \renewcommand*{\SR@endfoot}{\SR@@endfoot}%

425    \ifthenelse{\equal{\SR@caption}{\@empty}}%
426      {% CAPTION NOT GIVEN
427       \addtocounter{\LFcounter}{-1}%
428       \ifSR@continued%
429         \renewcommand*{\SR@conthead}{\SR@@captioncontinued%
430                       \SR@@captionskip\SR@@conthead}%
431      \fi%
432      }%
```

If a caption has not been specified, the first heading is empty. If the longfigure is a continuation (\SR@continued is true), the contination heading is set and the longfigure counter \LFcounter is decremented. Decrementing \LFcounter anticipates the automatic incrementing of the counter by the longfigure environment in the longfigure package. Thus, *continued* longfigures retain the number of the preceding longfigure environment.

```
433      {% CAPTION GIVEN
434       \renewcommand*{\SR@firsthead}{\SR@@caption\label{\SR@label}%
435                                     \SR@@captionskip\SR@@firsthead}%
436       \renewcommand*{\SR@conthead}{\SR@@captioncontinued%
437                                     \SR@@captionskip\SR@@conthead}%
```

```
438        }
439  }
```

If a caption has been specified, both the first heading and the continued heading are set appropriately.

\SR@set@outmargin The \SR@set@outmargin macro sets the longfigure parameter LFleft based on the output dimensions. The output width is set in the \Listing (listing destination), or \Boxgraphic macros. The length parameter \SR@scratchlength contains the output width.

```
440 \def\SR@set@outmargin{%
441     \ifdim\SR@scratchlength>\textwidth%
442       \dosloppy%
443       \setlength{\LFleft}{(\paperwidth-\SR@scratchlength)/\real{2.0}}%
444       \addtolength{\LFleft}{-1in}%
445       \addtolength{\LFleft}{-\hoffset}%
446       \addtolength{\LFleft}{-\leftskip}%
447       \ifodd\c@page\addtolength{\LFleft}{-\oddsidemargin}%
448       \else\addtolength{\LFleft}{-\evensidemargin}%
449       \fi%
450     \else%
451       \setlength{\LFleft}{(\textwidth-\SR@scratchlength)/\real{2.0}}%
452     \fi%
453 }
```

If the width of the output is narrower than the text block, it is centered with respect to the text block.

$$\mathsf{LFleft} = \frac{\texttt{textwidth} - \texttt{outputwidth}}{2}$$

If the width of the output is wider than the textblock, it is centered with respect to the page. The position of the left-hand edge of the page from the text block is calculated. The text block margin is the sum of the following:

- the standard TeX margin (1 inch)

- \hoffset

- \leftskip

- the margin specified in the document design

To move to the left edge of the page, move left by the total amount of margin:
1 inch + \hoffset + \leftskip + *document margin*

Therefore, to center with respect to the page, add horizontal space equal to half the difference between the paper width and the output width. That is:

$$\mathtt{LFleft} = -(\mathrm{TeXmargin} + \mathtt{hoffset} + \mathtt{leftskip} + \mathtt{margin}) + \frac{\mathrm{paperwidth} - \mathrm{outputwidth}}{2}$$

The first term (encompassed in parentheses) moves to the left edge of the page; the second and last term adds the horizontal space necessary to center the output. For example, using `letterpaper`, an output width of 5.75 inches, and a side margin of 1.5 inch, and assuming `\hoffset` and `\leftskip` have their default values of zero width, the left margin is set as follows:

$$\mathrm{LFleft} = -(1 + 1.5) + \frac{8.5 - 5.75}{2} = -1.125$$

The `\LFleft` margin is set to `-1.125`, which is 1.125 inches to the left of the normal text block margin.

For documents with unequal margins on even-numbered and odd-numbered pages, take the page number into account before subtracting the side margin. The `\ifodd` macro, provided by the `ifthen` package, checks whether the page number is odd or even.

Note: The `\dosloppy` command is used only when the output is wider than the text block.

\SR@insert The `\SR@insert` macro inserts an output stream into the document.

The following statements set the heading for the output and box the first output in the stream:

```
454 \newcommand{\SR@insert}[1]{%
455   \SR@set@outheadings%
456   \ifthenelse{\equal{\SR@type}{listing}}{\Boxlisting{#1}}{\Boxgraphic{#1}}%
457   \ifthenelse{\equal{\SR@dest}{latex}}%
458     {\relax}%
459     {\SR@set@outmargin}%
```

The width of a `\Listing` output is calculated in the `\Listing` macro. The width of a `\Graphic` output is calculated in the `\Boxgraphic` macro. Therefore, after boxing an output, the `\SR@scratchlength` contains the width of the output regardless of type. With that information, the margins that are required for centering are set. This does not apply to outputs with `dest=latex`.

The following statements determine whether an output file exists that matches the argument specified in the output tag; and if so, they set the Boolean switch SR@multifile to true and display the resulting boxed output:

```
460    \ifthenelse{\equal{\SR@insertname}{\@empty}}%
461        {\setboolean{SR@multifile}{false}}%
462        {\setboolean{SR@multifile}{true}}%
```

The following statements center a listing or a graphic and put a vertical line on each side of a listing (listing destination). No horizontal or vertical lines (no box) are drawn for graphics or outputs using the latex destination.

```
463    \ifthenelse{\equal{\SR@dest}{listing}}%
464      {\bgroup\longfigure{|c|}}{\bgroup\longfigure{c}}%
465       \SR@firsthead\SR@conthead\SR@endfoot%
466       \usebox{\SR@filebox}%
```

The following statements begin the input loop, inserting output files that match the specified stem pattern until there are no matching files:

```
467            \setcounter{SR@multifilecount}{1}%
468            \whiledo{\boolean{SR@multifile}}{%
469              \ifthenelse{\equal{\SR@type}{listing}}%
470                {\Boxlisting{#1\arabic{SR@multifilecount}}}%
471                {\Boxgraphic{#1\arabic{SR@multifilecount}}}%
472              \ifthenelse{\equal{\SR@insertname}{\@empty}}%
473                {\setboolean{SR@multifile}{false}}%
474                {\\[0.5\baselineskip]\usebox{\SR@filebox}%
475              \stepcounter{SR@multifilecount}%
476              \setboolean{SR@multifile}{true}}%
477            }%
478         \endlongfigure\egroup%
479      \unsloppy%
480 }
```

The logic of the preceding statements is depicted in Algorithm 4.

```
 1  while multiple files is True do
 2  │    if type=listing then
 3  │    │    Boxlisting
 4  │    else
 5  │    │    Boxgraphic
 6  │    end
 7  │    if file exists then
 8  │    │    insert contents, increment counter
 9  │    else
10  │    │    multiple files is False
11  │    end
12  end
```

Algorithm 4: Input Loop for the Output Stream

When SAS generates a set of files from one ODS selection, it follows a pattern. The first file that is generated is identical to the filename. The next file that is generated has the same name with a "1" appended to it, the next file has the same name with a "2" appended, and so on. The preceding statements input the original file; the following statements test for the existence of a file that has a name with the next digit in the sequence appended. The `\SR@multifilecount` counter contains the digit. With each inserted file, the counter is incremented.

When no file matches the pattern (the `\SR@insertname` macro is `\@empty`), the Boolean switch `\SR@multifile` is set to false and the loop is ended.

The `\unsloppy` macro is called when the stream insertion is finished to undo the effects of a `\dosloppy` command, which is called when the width of the output is greater than the width of the textblock. Invoking the `\unsloppy` macro multiple times has the same effect as calling it once; that is, even if `\dosloppy` is never called, calling `\unsloppy` causes no problem.

<*longfigure>

4 longfigure Usage

The `longfigure` package uses and relabels components of the well-known `longtable` package, written by David Carlisle, to provide a table-like environ-

ment that can display a stream of subfigures as a single figure that can break across pages.

The `longtable` package defines a `longtable` environment, which produces tables that can be broken by TEX's standard page-breaking algorithm. Similarly, the `longfigure` package defines a `longfigure` environment, which produces figures that can be broken by TEX's standard page-breaking algorithm. The internal structure of a long figure is similar to a long table. Rows might contain (for example) tables or graphics. Page breaks can occur only between rows.

The `longfigure` package differs from the `longtable` package in the following ways:

- The `longfigure` package supports two additional key-value options:

    - The `figname=` option specifies the counter for numbering `longfigure` environments. You can specify any string; the default is `figure`. When you specify a `figname=` value for which no counter already exists, the `longfigure` package loads the `tocloft` package and creates the counter.

    - The `resetby=` option specifies a counter (for example, `resetby=chapter`) such that output numbering is reset each time the counter value changes. If a counter is specified that does not exist, the `tocloft` package is loaded to create the new counter. For information about how the lists are typeset, see the `tocloft` package documentation.

- The counters and macros that start with \LT in the `longtable` package are renamed to start with \LF in the `longfigure` pacakge to avoid namespace conflicts when the two packages are used together. The generic macros that are defined in the `longtable` package (\endfirsthead, \endhead, \endfoot, and \endlastfoot) are also renamed with \LF as a prefix in the `longfigure` package.

- The \LF@name macro is based on the \fnum@table macro from the `longtable` package. The \LF@name macro returns the capitalized counter name and value. For example, if the counter is `figure` and the macro is processing the second `longfigure`, the \LF@name macro would contain the value "Figure 2."

You can use the `longfigure` package defaults to produce a *List of Figures* by inserting the following tag in your document at the point where you want the list to appear:

```
\listoffigures
```

The default counter used to display figures is the `figure` counter, but you can specify a different counter. For example, if you want your figures to be labeled as "Display," specify `figname=display` when you load the `longfigure` package; to display a *List of Displays*, insert the following command in your document at the point where you want the list to appear:

```
\listofdisplay
```

Note: If you specify a counter that does not exist, an auxiliary file with extension `.lft` is created to contain the information needed to create the list.

If you want to use more advanced features of the `tocloft` package, load it before you load the `longfigure` package so that the `longfigure` package sees that the counters specified by the `figname=` and `resetby=` options are already defined and does not attempt to create them.

4.1   Example

The following lines produce a single figure that contains three images and one tabular environment. Each element is a row of the `longfigure` environment. Page breaks can occur between rows.

```
\documentclass{book}
\usepackage{graphicx}
\usepackage{longfigure}

\begin{document}
  \begin{longfigure}{c}
    \caption{My Long Figure}\label{mlfig}\\
    \includegraphics[width=3in]{myfig1}\\
    \includegraphics[width=3in]{myfig2}\\
    \includegraphics[width=3in]{myfig3}\\
    \begin{tabular}{ll}
      one & two \\
      three & four\\
    \end{tabular}
  \end{longfigure}
\end{document}
```

In this example, the {c} argument in the \begin{longfigure} command speci-
fies only a single centered column. You can also specify multiple columns and, if
needed, use the \multicolumn command for more flexibility.

The following lines display another example that specifies a 'continued' heading
when the figure breaks over a page. It also displays a double horizontal line at the
end of the figure.

```
\documentclass{book}
\usepackage{graphicx}
\usepackage{longfigure}

\begin{document}
  \begin{longfigure}{c}
    \caption{My Long Figure}\label{mlfig2}\\
    \hline\endLFfirsthead
    \caption{continued}\\
    \hline\endLFhead
    \hline\endLFfoot
    \hline\hline\endLFlastfoot
    \includegraphics[width=3in]{myfig1}\\
    \includegraphics[width=3in]{myfig2}\\
    \includegraphics[width=3in]{myfig3}\\
    \begin{tabular}{ll}
      one & two \\
      three & four\\
    \end{tabular}
  \end{longfigure}
\end{document}
```

5   Implementation

This section describes the implementation of the longfigure package. The com-
ments describe only the changes from the longtable package code. For complete
details about the logic and usage of the longtable environment, see Carlisle
(2004).

481 \ProvidesPackage{longfigure}[2014/01/06 longfigure]

The following statement loads the xkeyval package for declaring and processing

43

package options:

```
482 \RequirePackage{xkeyval}
```

The following statement defines a new command, \LFcounter, to contain the string figure. Later code tests whether a counter with that name exists.

```
483 \newcommand*{\LFcounter}{figure}
```

The following statement defines a new command, \LFreset, to contain the name of the counter within which the longfigure number should reset. If no value is specified, the long figures are numbered consecutively through the document.

```
484 \newcommand*{\LFreset}{\@empty}
```

5.1 Options

The \LFcounter and \LFreset commands support the package options figname= and resetby= as follows:

```
485 \DeclareOptionX{figname}[figure]{\renewcommand*{\LFcounter}{#1}}
486 \DeclareOptionX{resetby}{\renewcommand*{\LFreset}{#1}}
```

The following statements further define the options that the longtable package defines:

```
487 \DeclareOptionX{set}{}
488 \DeclareOptionX{final}{}
489 \DeclareOptionX{errorshow}{\def\LF@warn{\PackageInfo{longfigure}}}
490 \DeclareOptionX{pausing}{\def\LF@warn#1{\LF@err{#1}{This is not really an error}}}
491 \ProcessOptionsX
```

The following statements process the options:

```
492 \def\LFProcessOptions#1{
493   \@ifundefined{c@#1}{%
494       \RequirePackage{tocloft}
495       \def\LFuc##1##2{\MakeUppercase{##1}{##2}}
496       \expandafter\def\csname list#1name\endcsname{List of \LFuc#1s}
497       \ifx\@empty\LFreset%
498         \newlistof{#1}{lft}{\csname list#1name\endcsname}
499       \else
500         \newlistof[\LFreset]{#1}{lft}{\csname list#1name\endcsname}
501       \fi
```

```
502    }{}%
503 }
504 \expandafter\LFProcessOptions\expandafter{\LFcounter}
```

If a counter is specified that does not exist, its name (\c@*countername*) is unde-fined and the `longfigure` package loads the `tocloft` package in order to use its commands to create the new counters and list.

Thus, the `tocloft` package is required only when a new counter is specified, and this automatic loading takes place only if the counter that is specified in the package options is not defined.

You can load the `tocloft` package before loading the `longfigure` package and retain all of the flexibility that the `tocloft` package offers. However, you must define the new counters yourself by using the `\newlistof` command in the `tocloft` package, and you must define the new list to use an auxiliary `lft` file where its auxiliary information is written.

5.2   Utilities

\strcfstr   The following macro, \strcfstr, checks whether two strings, which are pro-vided as arguments, are equal (Wilson, 2001). A new boolean \ifLF@same con-tains the result of the test.

```
505 \newif\ifLF@same
506 \newcommand{\strcfstr}[2]{%
507    \LF@samefalse
508    \begingroup\def\2{#2}
509      \ifx\2#1\endgroup\LF@sametrue
510      \else\endgroup
511      \fi
512 }
```

\LFupcase   The following macro, \LFupcase, uppercases the first letter of a string (Lazarides, 2010):

```
513 \def\LFupcase#1{%
514    \def\x##1##2{%
515      \MakeUppercase{##1}{##2}}\x#1%
516 }
```

The following macro, \LF@name, creates a string to provide a label and number for an output. Analogous to the \fnum@table macro in the `longtable` package,

45

it contains the capitalized version of the counter name and the counter number (for example, `Figure~3`).

```
517 \def\LF@name{\expandafter\LFupcase%
518            \expandafter{\LFcounter}~%
519            \expandafter\csname the\LFcounter\endcsname}%
```

The remainder of this package follows the `longtable` package almost identically, except that macros, skips, counters, and so on use an `\LF` prefix instead of the `\LT` prefix that the `longtable` package uses.

```
520 \def\LF@err{\PackageError{longfigure}}
521 \def\LF@warn{\PackageWarning{longfigure}}
522 \def\LF@final@warn{%
523   \AtEndDocument{%
524     \LF@warn{\LFcounter \@width s have changed. Rerun \LaTeX\.\@gobbletwo}}%
525   \global\let\LF@final@warn\relax}
526 %
527 \newskip\LFleft        \LFleft=\fill
528 \newskip\LFright       \LFright=\fill
529 \newskip\LFpre         \LFpre=\bigskipamount
530 \newskip\LFpost        \LFpost=\bigskipamount
531 \newcount\LFchunksize \LFchunksize=20
532 \let\c@LFchunksize\LFchunksize
533 \newdimen\LFcapwidth  \LFcapwidth=4in
534 \newbox\LF@head
535 \newbox\LF@firsthead
536 \newbox\LF@foot
537 \newbox\LF@lastfoot
538 \newcount\LF@cols
539 \newcount\LF@rows
540 \newcounter{LF@tables}
541 \newcounter{LF@chunks}[LF@tables]
542 %
543 \newtoks\LF@p@ftn
544 \mathchardef\LF@end@pen=30000
545 \def\longfigure{%
546   \par
547   \ifx\multicols\@undefined
548   \else
549     \ifnum\col@number>\@ne
550       \@twocolumntrue
551     \fi
552   \fi
553   \if@twocolumn
```

```
554    \LF@err{longfigure not in 1-column mode}\@ehc
555    \fi
556    \begingroup
557    \@ifnextchar[\LF@array{\LF@array[x]}}
558 \def\LF@array[#1]#2{%
559    \refstepcounter{\LFcounter}\stepcounter{LF@tables}%
560    \if l#1%
561       \LFleft\z@ \LFright\fill
562    \else\if r#1%
563       \LFleft\fill \LFright\z@
564    \else\if c#1%
565       \LFleft\fill \LFright\fill
566    \fi\fi\fi
567    \let\LF@mcol\multicolumn
568    \let\LF@@tabarray\@tabarray
569    \let\LF@@hl\hline
570    \def\@tabarray{%
571       \let\hline\LF@@hl
572       \LF@@tabarray}%
573    \let\\\LF@tabularcr\let\tabularnewline\\%
574    \def\newpage{\noalign{\break}}%
575    \def\pagebreak{\noalign{\ifnum`}=0\fi\@testopt{\LF@no@pgbk-}4}}%
576    \def\nopagebreak{\noalign{\ifnum`}=0\fi\@testopt\LF@no@pgbk4}}%
577    \let\hline\LF@hline \let\kill\LF@kill\let\caption\LF@caption
578    \@tempdima\ht\strutbox
579    \let\@endpbox\LF@endpbox
580    \ifx\extrarowheight\@undefined
581       \let\@acol\@tabacol
582       \let\@classz\@tabclassz \let\@classiv\@tabclassiv
583       \def\@startpbox{\vtop\LF@startpbox}%
584       \let\@@startpbox\@startpbox
585       \let\@@endpbox\@endpbox
586       \let\LF@LL@FM@cr\@tabularcr
587    \else
588       \advance\@tempdima\extrarowheight
589       \col@sep\tabcolsep
590       \let\@startpbox\LF@startpbox\let\LF@LL@FM@cr\@arraycr
591    \fi
592    \setbox\@arstrutbox\hbox{\vrule
593       \@height \arraystretch \@tempdima
594       \@depth \arraystretch \dp \strutbox
595       \@width \z@}%
596    \let\@sharp##\let\protect\relax
597     \begingroup
598       \@mkpream{#2}%
```

47

```
599    \xdef\LF@bchunk{%
600        \global\advance\c@LF@chunks\@ne
601        \global\LF@rows\z@\setbox\z@\vbox\bgroup
602        \LF@setprevdepth
603        \tabskip\LFleft \noexpand\halign to\hsize\bgroup
604        \tabskip\z@ \@arstrut \@preamble \tabskip\LFright \cr}%
605    \endgroup
606    \expandafter\LF@nofcols\LF@bchunk&\LF@nofcols
607    \LF@make@row
608    \m@th\let\par\@empty
609    \everycr{}\lineskip\z@\baselineskip\z@
610    \LF@bchunk}
611 \def\LF@no@pgbk#1[#2]{\penalty #1\@getpen{#2}\ifnum`{=0\fi}}
612 \def\LF@start{%
613    \let\LF@start\endgraf
614    \endgraf\penalty\z@\vskip\LFpre
615    \dimen@\pagetotal
616    \advance\dimen@ \ht\ifvoid\LF@firsthead\LF@head\else\LF@firsthead\fi
617    \advance\dimen@ \dp\ifvoid\LF@firsthead\LF@head\else\LF@firsthead\fi
618    \advance\dimen@ \ht\LF@foot
619    \dimen@ii\vfuzz
620    \vfuzz\maxdimen
621        \setbox\tw@\copy\z@
622        \setbox\tw@\vsplit\tw@ to \ht\@arstrutbox
623        \setbox\tw@\vbox{\unvbox\tw@}%
624    \vfuzz\dimen@ii
625    \advance\dimen@ \ht
626            \ifdim\ht\@arstrutbox>\ht\tw@\@arstrutbox\else\tw@\fi
627    \advance\dimen@\dp
628            \ifdim\dp\@arstrutbox>\dp\tw@\@arstrutbox\else\tw@\fi
629    \advance\dimen@ -\pagegoal
630    \ifdim \dimen@>\z@\vfil\break\fi
631        \global\@colroom\@colht
632    \ifvoid\LF@foot\else
633        \advance\vsize-\ht\LF@foot
634        \global\advance\@colroom-\ht\LF@foot
635        \dimen@\pagegoal\advance\dimen@-\ht\LF@foot\pagegoal\dimen@
636        \maxdepth\z@
637    \fi
638    \ifvoid\LF@firsthead\copy\LF@head\else\box\LF@firsthead\fi\nobreak
639    \output{\LF@output}}
640 \def\endlongfigure{%
641    \crcr
642    \noalign{%
643        \let\LF@entry\LF@entry@chop
```

48

```
644      \xdef\LF@save@row{\LF@save@row}}%
645    \LF@echunk
646    \LF@start
647    \unvbox\z@
648    \LF@get@widths
649    \if@filesw
650      {\let\LF@entry\LF@entry@write\immediate\write\@auxout{%
651        \gdef\expandafter\noexpand
652          \csname LF@\romannumeral\c@LF@tables\endcsname
653            {\LF@save@row}}}%
654    \fi
655    \ifx\LF@save@row\LF@@save@row
656    \else
657      \LF@warn{Column \@width s have changed\MessageBreak
658              in table \thetable}%
659      \LF@final@warn
660    \fi
661    \endgraf\penalty -\LF@end@pen
662    \endgroup
663    \global\@mparbottom\z@
664    \pagegoal\vsize
665    \endgraf\penalty\z@\addvspace\LFpost
666    \ifvoid\footins\else\insert\footins{}\fi}
667 \def\LF@nofcols#1&{%
668    \futurelet\@let@token\LF@n@fcols}
669 \def\LF@n@fcols{%
670    \advance\LF@cols\@ne
671    \ifx\@let@token\LF@nofcols
672      \expandafter\@gobble
673    \else
674      \expandafter\LF@nofcols
675    \fi}
676 \def\LF@tabularcr{%
677    \relax\iffalse{\fi\ifnum0=`}\fi
678    \@ifstar
679      {\def\crcr{\LF@crcr\noalign{\nobreak}}\let\cr\crcr
680       \LF@t@bularcr}%
681      {\LF@t@bularcr}}
682 \let\LF@crcr\crcr
683 \let\LF@setprevdepth\relax
684 \def\LF@t@bularcr{%
685    \global\advance\LF@rows\@ne
686    \ifnum\LF@rows=\LFchunksize
687      \gdef\LF@setprevdepth{%
688        \prevdepth\z@\global
```

```
689        \global\let\LF@setprevdepth\relax}%
690      \expandafter\LF@xtabularcr
691    \else
692      \ifnum0='{}\fi
693      \expandafter\LF@LL@FM@cr
694    \fi}
695 \def\LF@xtabularcr{%
696    \@ifnextchar[\LF@argtabularcr\LF@ntabularcr}
697 \def\LF@ntabularcr{%
698    \ifnum0='{}\fi
699    \LF@echunk
700    \LF@start
701    \unvbox\z@
702    \LF@get@widths
703    \LF@bchunk}
704 \def\LF@argtabularcr[#1]{%
705    \ifnum0='{}\fi
706    \ifdim #1>\z@
707      \unskip\@xargarraycr{#1}%
708    \else
709      \@yargarraycr{#1}%
710    \fi
711    \LF@echunk
712    \LF@start
713    \unvbox\z@
714    \LF@get@widths
715    \LF@bchunk}
716 \def\LF@echunk{%
717    \crcr\LF@save@row\cr\egroup
718    \global\setbox\@ne\lastbox
719      \unskip
720    \egroup}
721 \def\LF@entry#1#2{%
722    \ifhmode\@firstofone{&}\fi\omit
723    \ifnum#1=\c@LF@chunks
724    \else
725      \kern#2\relax
726    \fi}
727 \def\LF@entry@chop#1#2{%
728    \noexpand\LF@entry
729      {\ifnum#1>\c@LF@chunks
730         1}{0pt%
731      \else
732        #1}{#2%
733      \fi}}
```

```
734 \def\LF@entry@write{%
735   \noexpand\LF@entry^^J%
736   \@spaces}
737 \def\LF@kill{%
738   \LF@echunk
739   \LF@get@widths
740   \expandafter\LF@rebox\LF@bchunk}
741 \def\LF@rebox#1\bgroup{%
742   #1\bgroup
743   \unvbox\z@
744   \unskip
745   \setbox\z@\lastbox}
746 \def\LF@blank@row{%
747   \xdef\LF@save@row{\expandafter\LF@build@blank
748     \romannumeral\number\LF@cols 001 }}
749 \def\LF@build@blank#1{%
750   \if#1m%
751     \noexpand\LF@entry{1}{0pt}%
752     \expandafter\LF@build@blank
753   \fi}
754 \def\LF@make@row{%
755   \global\expandafter\let\expandafter\LF@save@row
756     \csname LF@\romannumeral\c@LF@tables\endcsname
757   \ifx\LF@save@row\relax
758     \LF@blank@row
759   \else
760     {\let\LF@entry\or
761     \if!%
762        \ifcase\expandafter\expandafter\expandafter\LF@cols
763        \expandafter\@gobble\LF@save@row
764        \or
765        \else
766          \relax
767        \fi
768      !%
769     \else
770       \aftergroup\LF@blank@row
771     \fi}%
772   \fi}
773 \let\setlongfigures\relax
774 \def\LF@get@widths{%
775   \setbox\tw@\hbox{%
776     \unhbox\@ne
777     \let\LF@old@row\LF@save@row
778     \global\let\LF@save@row\@empty
```

```
779      \count@\LF@cols
780      \loop
781        \unskip
782        \setbox\tw@\lastbox
783      \ifhbox\tw@
784        \LF@def@row
785        \advance\count@\m@ne
786      \repeat}%
787  \ifx\LF@@save@row\@undefined
788    \let\LF@@save@row\LF@save@row
789  \fi}
790  \def\LF@def@row{%
791    \let\LF@entry\or
792    \edef\@tempa{%
793      \ifcase\expandafter\count@\LF@old@row
794      \else
795        {1}{0pt}%
796      \fi}%
797    \let\LF@entry\relax
798    \xdef\LF@save@row{%
799      \LF@entry
800      \expandafter\LF@max@sel\@tempa
801      \LF@save@row}}
802  \def\LF@max@sel#1#2{%
803    {\ifdim#2=\wd\tw@
804        #1%
805      \else
806        \number\c@LF@chunks
807      \fi}%
808    {\the\wd\tw@}}
809  \def\LF@hline{%
810    \noalign{\ifnum0='}\fi
811      \penalty\@M
812      \futurelet\@let@token\LF@@hline}
813  \def\LF@@hline{%
814    \ifx\@let@token\hline
815      \global\let\@gtempa\@gobble
816      \gdef\LF@sep{\penalty-\@medpenalty\vskip\doublerulesep}%
817    \else
818      \global\let\@gtempa\@empty
819      \gdef\LF@sep{\penalty-\@lowpenalty\vskip-\arrayrulewidth}%
820    \fi
821    \ifnum0='{\fi}%
822    \multispan\LF@cols
823      \unskip\leaders\hrule\@height\arrayrulewidth\hfill\cr
```

```
824    \noalign{\LF@sep}%
825    \multispan\LF@cols
826        \unskip\leaders\hrule\@height\arrayrulewidth\hfill\cr
827    \noalign{\penalty\@M}%
828    \@gtempa}
```

5.3 Captioning

You can easily change how a long figure is captioned by redefining the `\LF@makecaption`
macro after loading the `longfigure` package. The following statements show the
default definition of the `\LF@makecaption`:

```
829 \def\LF@caption{%
830    \noalign\bgroup
831        \@ifnextchar[{\egroup\LF@c@ption\@firstofone}\LF@capti@n}
```

The `\LF@caption` command begins the process. If it includes an optional argu-
ment, it calls `\LF@c@ption`; otherwise it calls `\LF@capti@n`, which then calls
`\LF@c@ption`.

```
832 \def\LF@c@ption#1[#2]#3{%
833    \LF@makecaption#1\LF@name{#3}%
834    \def\@tempa{#2}%
835    \ifx\@tempa\@empty\else
```

If a list of long figures is requested, the following code uses the previously defined
`\strcfstr` macro and `\ifLF@same` boolean to determine the name of the counter
and set the output file to contain the `longfigure` information.

The code writes to one of the following files.

- If the counter is `figure`, write to the `lof` file.

- If the counter is `table`, write to the `lot` file.

- Otherwise, write to `lft`, a file created here for this purpose.

```
836        {\let\\\space
837        \strcfstr{\LFcounter}{figure}
838          \ifLF@same\def\LFoutfile{lof}\else
839            \strcfstr{\LFcounter}{table}
840              \ifLF@same\def\LFoutfile{lot}\else
841                \def\LFoutfile{lft}\fi\fi
842        \addcontentsline{\LFoutfile}{\LFcounter}
843        {\expandafter\protect\expandafter\numberline\expandafter%
```

```
844    {\expandafter\csname the\LFcounter\endcsname}{#2}}}%
845    \fi
846 }
```

The `\LF@c@ption` macro ends the process when it calls the `\LF@makecaption` macro, which typesets the caption.

```
847 \def\LF@capti@n{%
848    \@ifstar
849      {\egroup\LF@c@ption\@gobble[]}%
850      {\egroup\@xdblarg{\LF@c@ption\@firstofone}}}
```

If you want to redefine how the `longfigure` is captioned, you need to override the following macro. The first argument is the name of the counter (for example, `Figure`), the second argument is the number of the counter, and the third argument is the caption itself.

```
851 \def\LF@makecaption#1#2#3{%
852   \LF@mcol\LF@cols c{\hbox to\z@{\hss\parbox[t]\LFcapwidth{%
853     \sbox\@tempboxa{#1{#2: }#3}%
854     \ifdim\wd\@tempboxa>\hsize
855       #1{#2: }#3%
856     \else
857       \hbox to\hsize{\hfil\box\@tempboxa\hfil}%
858     \fi
859     \endgraf\vskip\baselineskip}%
860   \hss}}}
861 \def\LF@output{%
862   \ifnum\outputpenalty <-\@Mi
863     \ifnum\outputpenalty > -\LF@end@pen
864       \LF@err{floats and marginpars not allowed in a longfigure}\@ehc
865     \else
866       \setbox\z@\vbox{\unvbox\@cclv}%
867       \ifdim \ht\LF@lastfoot>\ht\LF@foot
868         \dimen@\pagegoal
869         \advance\dimen@-\ht\LF@lastfoot
870         \ifdim\dimen@<\ht\z@
871           \setbox\@cclv\vbox{\unvbox\z@\copy\LF@foot\vss}%
872           \@makecol
873           \@outputpage
874           \setbox\z@\vbox{\box\LF@head}%
875         \fi
876       \fi
877       \global\@colroom\@colht
878       \global\vsize\@colht
```

```
879        \vbox
880          {\unvbox\z@\box\ifvoid\LF@lastfoot\LF@foot\else
881           \LF@lastfoot\fi}%
882        \fi
883    \else
884      \setbox\@cclv\vbox{\unvbox\@cclv\copy\LF@foot\vss}%
885      \@makecol
886      \@outputpage
887        \global\vsize\@colroom
888      \copy\LF@head\nobreak
889    \fi}
890 \def\LF@end@hd@ft#1{%
891    \LF@echunk
892    \ifx\LF@start\endgraf
893      \LF@err
894        {Longfigure head or foot not at start of table}%
895        {Increase LFchunksize}%
896    \fi
897    \setbox#1\box\z@
898    \LF@get@widths
899    \LF@bchunk}
```

The following four macros do not have an \LT prefix in the longtable package, but they must be redefined to have an \LF prefix in order to avoid a namespace clash;

```
900 \def\endLFfirsthead{\LF@end@hd@ft\LF@firsthead}
901 \def\endLFhead{\LF@end@hd@ft\LF@head}
902 \def\endLFfoot{\LF@end@hd@ft\LF@foot}
903 \def\endLFlastfoot{\LF@end@hd@ft\LF@lastfoot}
904 %
905 \def\LF@startpbox#1{%
906    \bgroup
907      \let\@footnotetext\LF@p@ftntext
908      \setlength\hsize{#1}%
909      \@arrayparboxrestore
910      \vrule \@height \ht\@arstrutbox \@width \z@}
911 \def\LF@endpbox{%
912    \@finalstrut\@arstrutbox
913    \egroup
914    \the\LF@p@ftn
915    \global\LF@p@ftn{}%
916    \hfil}
917 \def\LF@p@ftntext#1{%
918    \edef\@tempa{\the\LF@p@ftn\noexpand\footnotetext[\the\c@footnote]}%
```

919   `\global\LF@p@ftn\expandafter{\@tempa{#1}}}%`

5.4 References

Carlisle, D. 2004. *The longtable Package*. Included in the "Comprehensive TeX Archive Network." http://ctan.org.

Lazarides, Y. 2010. TeXstackexchange, online forum. http://tex.stackexchange.com/questions/7992.

Schöpf, R, B. Raichle, and C. Rowley. 2001. *A New Implementation of LaTeX's* `verbatim` *and* `verbatim*` *Environments*. Originally appeared in TUGboat 1990, 11(2), 284–296.

Thanh, H., S. Rahtz, H. Hagen, and H. Henkel. 2009. "The pdfTeX User's Manual," Revision 655, corresponding to pdfTeX 1.40.11. www.tug.org/applications/pdftex.

Wilson, P. 2001. *Glisterings*. In TUGboat 22(4), 339–340.

6 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

58