

# The statrep package\*

Tim Arnold and Warren F. Kuhfeld  
SAS Institute Inc.  
tim.arnold@sas.com  
warren.kuhfeld@sas.com

2012/3/15

## Contents

<b>1</b>	<b>About This Document</b>	<b>2</b>
<b>2</b>	<b>StatRep Usage</b>	<b>2</b>
<b>3</b>	<b>StatRep Implementation</b>	<b>4</b>
3.1	Required Packages . . . . .	4
3.2	Programming Utilities . . . . .	4
3.3	Customizable Settings . . . . .	8
3.4	Code Environments . . . . .	13
3.4.1	Datastep Environment . . . . .	14
3.4.2	Sascode Environment . . . . .	19
3.5	Handling the SAS Output . . . . .	27
3.5.1	The \Listing Macro . . . . .	29
3.5.2	The \Graphic Macro . . . . .	30
3.5.3	The \Boxlisting Macro . . . . .	31
3.5.4	The \Boxgraphic Macro . . . . .	32
3.5.5	The \SR@insert Macro . . . . .	33
<b>4</b>	<b>longfigure Usage</b>	<b>38</b>
<b>5</b>	<b>longfigure Implementation</b>	<b>39</b>
5.1	Options . . . . .	40

---

\*This document corresponds to statrep v1.0, last revised 2012/3/15.

5.2	Utilities	41
5.3	Captioning	48
5.4	References	51
6	Index	52

## List of Figures

1	Example of Using the StatRep Package	2
2	Processing Schematic for the Datastep Environment	15
3	Processing Schematic for the Sascode Environment	25

## 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 L<sup>A</sup>T<sub>E</sub>X 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  $\TeX$ Live 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{figname}{\PassOptionsToPackage{\CurrentOption}{longfigure}}
9 \DeclareOptionX{resetby}{\PassOptionsToPackage{\CurrentOption}{longfigure}}
10 \ProcessOptionsX
11 \RequirePackage{longfigure}
```

### 3.2 Programming Utilities

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

```
12 \ifSR@generate\def\SR@writepgm{\immediate\write}
13 \else\let\SR@writepgm\@gobbletwo
14 \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 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:

```
15 \newcommand*{\dosloppy}{\setlength{\hfuzz}{\maxdimen}\hbadness\maxdimen}
16 \newcommand*{\unsloppy}{\setlength{\hfuzz}{0.3pt}\hbadness 1414}
```

`\dosloppy`    The `dosloppy` macro is intended to be used when you knowingly violate type-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 L<sup>A</sup>T<sub>E</sub>X's line-breaking algorithms or line penalties; they suppress the overfull box (`\hfuzz`) and underfull box (`\hbadness`) warnings in the log. The system uses these commands when it insert outputs. See the `\SR@set@outmargin` macro in section 3.5.5 for details.

The following definitions are used to calculate and set the output stream of listings or figures. See section 3.5.5 for details.

```
17 \newlength{\SR@scratchlength}
18 \newlength{\SR@verbwidth}
19 \newsavebox{\SR@filebox}
20 \newcommand*{\SR@firsthead}{}
21 \newcommand*{\SR@conthead}{}
22 \newcommand*{\SR@endfoot}{}

```

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

```
23 \newcounter{SR@currentline}\setcounter{SR@currentline}{0}
24 \newcounter{SR@displayline}\setcounter{SR@displayline}{0}
25 \newcounter{SR@programline}\setcounter{SR@programline}{0}
26 \newcounter{SR@startinglastline}\setcounter{SR@startinglastline}{0}
27 \newcounter{SR@totallines}\setcounter{SR@totallines}{0}
28 \newcounter{SR@multifilecount}\setcounter{SR@multifilecount}{0}
29 \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 3.5.5 for complete details.

```
30 \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  $\text{\LaTeX}$  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.

```
31 \define@boolkeys{SR}{SR@}{display,program,continued}[true]{}
32 \define@cmdkeys{SR}{SR@}{caption,first,last,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 L<sup>A</sup>T<sub>E</sub>X 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.

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

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:

```
34 \define@cmdkeys{SR}[SR@]{style,dpi,firstobj,lastobj,objects,pattern}{}
```

```
35 \define@cmdkeys{SR}[SR@]{options,height,pagesize,type}{}
```

- `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 `presetekeys` macro, which provides default values for each key:

```
36 \presetkeys{SR}{%
37   display = false,   program = false,   continued = false,
38   caption = \@empty, first   = 0,       last       = 0,
39   fontsize = \@empty, scale   = \@empty, label      = \@empty,
40   %
41   store    = \@empty, linesize = \@empty, width      = \@empty,
42   %
43   type     = \@empty, style    = \@empty, dpi        = \@empty,
44   firstobj = \@empty, lastobj  = \@empty, objects    = \@empty,
45   pattern  = \@empty, options  = \@empty, height     = \@empty,
46   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.

```
47 %%%%%%%%%%%
48 %% Edit this line to point to the location of the StatRep macros.
49 \def\SRmacropath{statrep_macros.sas}
50 %%%%%%%%%%%
```

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

```
\SRlinesize
\SRpagesize
  \SRdpi
  \SRstyle
51 \def\SRlinesize{80}
52 \def\SRpagesize{500}
53 \def\SRdpi{300}
54 \def\SRstyle{statistical}
```

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.

```
55 \def\SRgraphicdir{png}
56 \def\SRlistingdir{lst}
```

```
\SRgraphicdir
\SRlistingdir
```

- \SRgraphicdir specifies the name of the directory that contains the graphic output files that are generated by SAS. 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.

```
57 \def\SRprogramname{\jobname_SR.sas}
58 \def\SRpreamblename{\jobname_SR_preamble.sas}
59 \def\SRmacroinclude{\@percentchar include "\SRmacropath" /nosource;}
```

```

60 \def\SRsasprogramline{x "del
61     \@percentchar nrstr(\@percentchar\@percentchar)CD%
62     \@percentchar nrstr(\@percentchar\@percentchar)%
63     \@backslashchar&listingdir\@backslashchar*.lst
64     \@percentchar nrstr(\@percentchar\@percentchar)CD%
65     \@percentchar nrstr(\@percentchar\@percentchar)%
66     \@backslashchar&graphicdir\@backslashchar*.png";
67 }

```

\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 specifies statements that remove the contents of the listing and graphic directories. The directories are created with each SAS run that includes the macros themselves (via x commands).

```

x "del %nrstr(%)CD%nrstr(%)&listingdir\*.lst
    %nrstr(%)CD%nrstr(%)&graphicdir\*.png";

```

```

68 \def\SRparindent{3em}
69 \def\SRintertext{... more data lines ...}
70 \def\SRtempfilename{sr.tmp}
71 \def\SRcontinuedname{continued}
72 \def\SRcaptionfont{\sfamily}
73 \def\SRcaptioncontinuedfont{\sfamily\itshape}
74 \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`.
- `\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:

```
75 \InputIfFileExists{statrep.cfg}
76 {\PackageInfo{statrep}{Reading custom configuration file statrep.cfg}}
77 {\PackageInfo{statrep}{No custom configuration file found (statrep.cfg).%
78                          Using defaults.}}
```

Two output streams are created:

```
79 \newwrite\SR@program@stream
80 \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:

```
81 \def\SR@preambletext{/*^^J
82   This file is auto-generated by the statrep package.^^J
83   Do not edit this file or your changes will be lost.^^J
84   Edit the LaTeX file instead.^^J
85   ^^J
86   See the statrep package documentation and the file^^J
```

```

87 statrep.cfg for information on these settings.^^J
88 */^^J
89 ^^J
90 }

```

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.

```

91 \def\SR@write@preamble{
92   \immediate\openout\SR@tempfile@stream\SRpreamblename%
93   \SR@writepgm\SR@tempfile@stream{\SR@preambletext}
94   \SR@writepgm\SR@tempfile@stream{/* Set and invoke macro variable defaults. */}
95   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultlinesize=\SRlinesize;}
96   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultpagesize=\SRpagesize;}
97   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultdpi=\SRdpi;}
98   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultstyle=\SRstyle;}
99   \SR@writepgm\SR@tempfile@stream{\@percentchar let listingdir=\SRlistingdir;}
100  \SR@writepgm\SR@tempfile@stream{\@percentchar let graphicdir=\SRgraphicdir;}
101  \SR@writepgm\SR@tempfile@stream{}
102  \SR@writepgm\SR@tempfile@stream{options noxwait nodate nonumber}
103  \SR@writepgm\SR@tempfile@stream{ls=&defaultlinesize ps=&defaultpagesize}
104  \SR@writepgm\SR@tempfile@stream{formchar='|----|+|---+=|-/\'@backslashchar<>*' ;}
105  \SR@writepgm\SR@tempfile@stream{}
106  \SR@writepgm\SR@tempfile@stream{ x "mkdir &graphicdir &listingdir";}
107  \SR@writepgm\SR@tempfile@stream{/* Include SAS macro definitions. */}
108  \SR@writepgm\SR@tempfile@stream{\SRmacroinclude}
109  \SR@writepgm\SR@tempfile@stream{}
110  \immediate\closeout\SR@tempfile@stream

```

The following statements begin writing the generated SAS program \SR@programname. 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 L<sup>A</sup>T<sub>E</sub>X source to a SAS session can include only the preamble in their SAS session.

```

111 \immediate\openout\SR@program@stream\SR@programname
112 \SR@writepgm\SR@program@stream{\SR@preambletext}
113 \SR@writepgm\SR@program@stream{\@percentchar include "\SRpreamblename" /nosource;}
114 \SR@writepgm\SR@program@stream{/* Remove all output files. */}
115 \SR@writepgm\SR@program@stream{\SRsasprogramline}
116 \SR@writepgm\SR@program@stream{}
117 \SR@writepgm\SR@program@stream{/* Start program with a null title. */}

```

```

118 \SR@writepgm\SR@program@stream{title;}
119 \SR@writepgm\SR@program@stream{}
120 }

```

`\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, `\SR@linesize`, `\SR@dpi`). You can modify the preamble in the `statrep.cfg` file.

```

121 \AtBeginDocument{\SR@write@preamble}
122 \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@program@stream` 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:

```

123 \def\SR@datastep@writefile#1{%
124   \setcounter{SR@totallines}{0}%
125   \@bsphack\immediate\openout\SR@tempfile@stream#1%
126   \let\do\@makeother\dospecials\catcode'\^M\active%
127   %
128   \def\verbatim@processline{%
129     \addtocounter{SR@totallines}{1}%
130     \SR@writepgm\SR@tempfile@stream{\the\verbatim@line}%
131   }%
132   \verbatim@start%
133 }
```

`\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).

```

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

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

```

135 \newcommand{\SR@datastep@writeprogramline}{%
136   \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 specified 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 redefining the `\verbatim@processline` macro, this time to process lines from the temporary file according to options that are specified in the Datastep environment.

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.

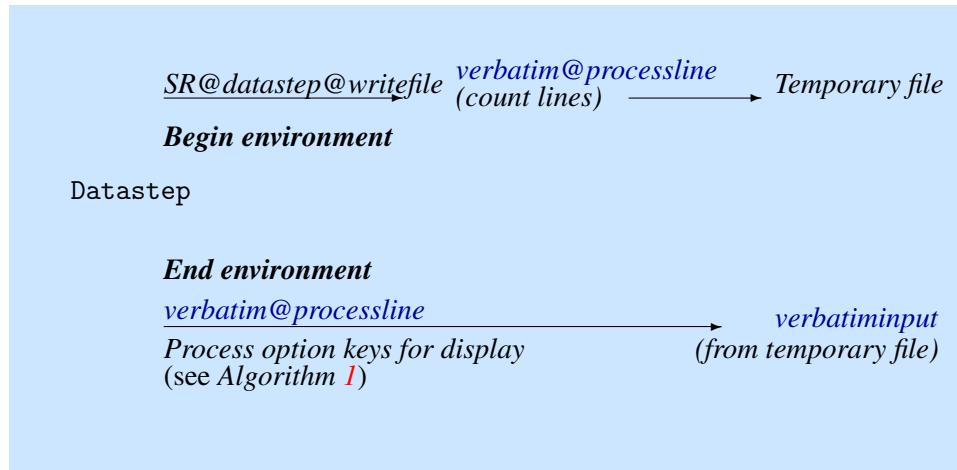


Figure 2: Processing Schematic for the Datastep Environment

```

137 \def\@Datastep[#1]{%
138   \setkeys{SR}{#1}%
139   \SR@datastep@writefile{\SRtempfilename}%
140 }
141 \newenvironment{Datastep}{%
142   \catcode'\^M=\active
143   \@ifnextchar[{\catcode'\^M=5\@Datastep}{\catcode'\^M=5\@Datastep[]}]
144 }

```

**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 possibility, 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.

```

145 {%

```

```
146 \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.

```
147 \ifSR@program\def\verbatim@processline{\SR@datastep@writeprogramline}%
148 \else%
```

Otherwise, the code block is to be displayed.

```
149 \ifthenelse{\equal{\SR@fontsize}{\@empty}}%
150 {\relax}%
151 {\@nameuse{\SR@fontsize}}%
152 \setcounter{SR@currentline}{0}%
153 \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.

```
154 \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.

```
155 \ifSR@display\else\SR@datastep@writeprogramline\fi
```

The remainder of the macro logic is described in Algorithm 1 on page 18.

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:

```

156      \ifnum\SR@first>0%
157      \ifnum\theSR@currentline<\SR@first%
158      {\hskip\SRparindent\the\verbatim@line\par}%
159      \else%
160      \ifnum\theSR@currentline=\SR@first%
161      \par\hskip\SRparindent\SRintertext\par%
162      \else%
163      \ifnum\SR@last>0%
164      \ifnum\theSR@currentline=\theSR@startinglastline\par\fi%
165      \ifnum\theSR@currentline>\theSR@startinglastline%
166      {\hskip\SRparindent\the\verbatim@line\par}%
167      \fi%
168      \fi%
169      \fi%
170      \fi%
171      \else%
172      \hskip\SRparindent\the\verbatim@line\par%
173      \fi%
174      \addtocounter{SR@currentline}{1}%
175      }%
176  \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      |
15    |
16  else
17    | display line;
18  |
19 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.

```
177 \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:

```

178 \ifSR@program
179 \setlength{\SR@scratchlength}{-2\topsep-\partopsep-2\parskip-2\baselineskip}
180 \vspace*{\SR@scratchlength}%
181 \fi%

```

If the block is written to the program file, the following statements output a blank line after this Datasstep environment in the program file:

```
182  \ifSR@display\else%
183    \SR@writepgm\SR@program@stream{}}%
184  \fi%
185 }
```

### 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).

```
186 \begingroup
187  \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.

```
188  \gdef\SR@sascode@addtoline{\catcode'\%12
189    \expandafter\relax
190    \expandafter\SR@sascode@filter
191    \expandafter{\number\catcode'\% }}
```

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 pdfL<sup>A</sup>T<sub>E</sub>X: 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 pdfT<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X. 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 L<sup>A</sup>T<sub>E</sub>X, 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 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

```

192 \gdef\SR@sascode@filter#1#2{
193   \ifnum\pdfmatch{~%\string\*}
194     {\detokenize{#2}}=1
195     \ifnum\pdfmatch{~%\string\*\space*program\space*([0-9]+);}
196       {\detokenize{#2}}=1
197       \setcounter{SR@programline}
198       {\expandafter\strip@prefix\pdflastmatch 1}
199       \catcode'\%#1\@tempswafalse\verbatim@line{}
200   \else
201     \ifnum\pdfmatch{~%\string\*\space*display\space*([0-9]+);}
202       {\detokenize{#2}}=1
203       \setcounter{SR@displayline}
204       {\expandafter\strip@prefix\pdflastmatch 1}
205       \catcode'\%#1\@tempswafalse\verbatim@line{}
206   \else
207     \ifnum\pdfmatch{~%\string\*\space*;(.*)$}
208       {\detokenize{#2}}=1
209       \SR@writepgm\SR@program@stream{\expandafter\strip@prefix\pdflastmatch 1}
210       \catcode'\%#1\@tempswafalse\verbatim@line{}

```

```

211         \else \catcode'\%#1\SR@sascode@writeline{#2}
212         \fi
213     \fi
214 \fi
215 \else
216     \catcode'\%#1\SR@sascode@writeline{#2}
217 \fi
218 }
219 \endgroup

```

The `\beginngroup` 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 (`\@tempwattrue`). 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 command, 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 program 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 environment display matches the `Datastep` environment display.

```

220 \newcommand{\SR@sascode@writeline}[1]{%
221   \ifSR@program%

```

```

222     \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
223     \@tempswafalse%
224 \else%
225     \ifSR@display%
226         \verbatim@line\expandafter{\the\verbatim@line\hskip\SR@parindent#1}%
227     \else%
228         \ifnum\theSR@programline>0%
229             \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
230             \addtocounter{SR@programline}{-1}%
231             \@tempswafalse%
232         \else%
233             \ifnum\theSR@displayline>0%
234                 \verbatim@line\expandafter{\the\verbatim@line\hskip\SR@parindent#1}%
235                 \addtocounter{SR@displayline}{-1}%
236             \else%
237                 \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
238                 \verbatim@line\expandafter{\the\verbatim@line\hskip\SR@parindent#1}%
239             \fi%
240         \fi%
241     \fi%
242 \fi%
243 }

```

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.



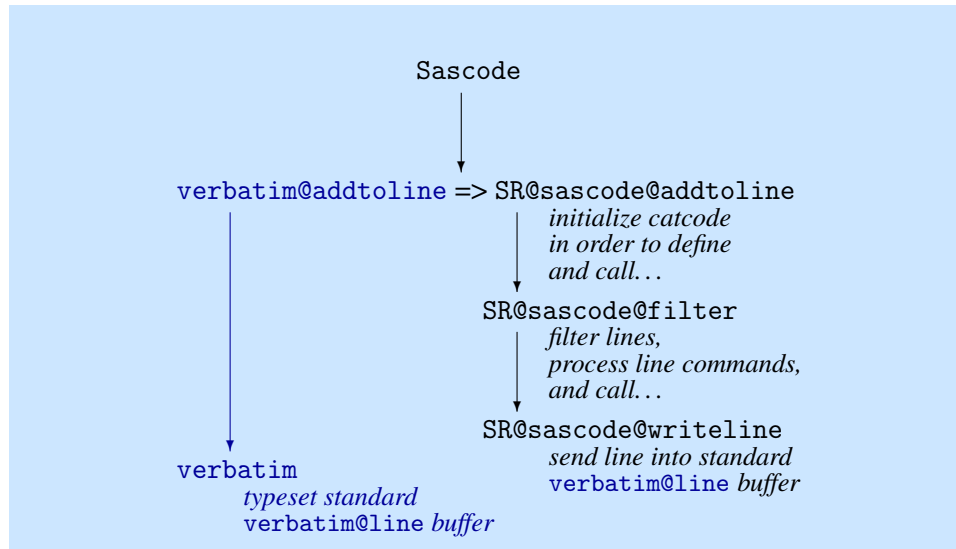


Figure 3: Processing Schematic for the Sascode Environment

**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 possibility, 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.

```

244 \def\@Sascode[#1]{%
245   \setkeys{SR}{#1}%
246   \ifSR@display\else%
247     \ifthenelse{\equal{\SR@store}{\@empty}}{%
248       {\relax}%
249       {\SR@writepgm\SR@program@stream{\@percentchar output(\SR@store)}}%
250       \PackageInfo{statrep}{Processing Sascode block \SR@store}}%
251   \fi%
252   \ifthenelse{\equal{\SR@fontsize}{\@empty}}{%
253     {\relax}%
254     {\@nameuse{\SR@fontsize}}}%
255   \setcounter{SR@programline}{0}%
256   \setcounter{SR@displayline}{0}%
257   \let\verbatim@addtoline\SR@sascode@addtoline%
258   \verbatim}
259 \newenvironment{Sascode}{%
260   \catcode'\^M=\active
261   \@ifnextchar[{\catcode'\^M=5\@Sascode}{\catcode'\^M=5\@Sascode[]}]
262 }
```

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:

```

263 {\endverbatim%
264   \setlength{\SR@scratchlength}{-2\topsep-\partopsep-2\parskip}
265   \vspace*{\SR@scratchlength}%
266   \ifSR@program\vspace*{-2\baselineskip}\fi
```

```

267 %
268 \ifSR@display\else%
269 \ifthenelse{\equal{\SR@store}{\@empty}}{%
270 {\relax}%
271 {\SR@writepgm\SR@program@stream{\@percentchar endoutput(\SR@store)}}%
272 \SR@writepgm\SR@program@stream{}}%
273 \fi%
274 }

```

### 3.5 Handling the SAS Output

The `\Listing` and `\Graphic` tags support options that are used in the generated program and in output display. 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 centered. Listing output is inserted in the same way, but the output is framed. The frame is created with the help of the `longfigure` package and the `\hline` command.

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.

```

275 \def\SR@write@outoptions{

```

```

276 \SR@set@outoptions
277 \let\#\SR@hashchar
278 \SR@writepgm\SR@program@stream{%
279 \@percentchar write(\SR@label,store=\SR@store \SR@final@outoptions)
280 }

```

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

```

281 \SR@writepgm\SR@program@stream{}
282 }

```

`\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.

```

283 \newcommand*{\SR@set@outoptions}{%
284 \global\edef\SR@final@outoptions{}
285 \SR@outkeyval{height}\SR@outkeyval{objects}\SR@outkeyval{pattern}
286 \SR@outkeyval{style}\SR@outkeyval{dpi}\SR@outkeyval{options}
287 \SR@outkeyval{pagesize}\SR@outkeyval{linesize}\SR@outkeyval{width}
288 \SR@outkeyval{type}\SR@outkeyval{firstobj}\SR@outkeyval{lastobj}
289 }

```

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.

```

290 \newcommand*{\SR@outkeyval}[1]{%
291 \def\SR@outkey{#1}
292 \edef\SR@outval{\csname SR@#1\endcsname}
293 \ifx\@empty\SR@outval
294 \else
295 \edef\SR@final@outoptions{\SR@final@outoptions,\SR@outkey=\SR@outval}
296 \fi
297 }

```

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 The `\Listing` Macro

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

```
298 \newcommand{\Listing}[2] [] {%
299   \setkeys{SR}{#1,type=listing,label=#2}
300   \ifthenelse{\equal{\SR@store}{\@empty}}
301     {\relax}
302     {\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.

The following statements set the font size:

```
303   \ifthenelse{\equal{\SR@fontsize}{\@empty}}
304     {\def\SR@fontsize{normalsize}}
305     {\relax}
```

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

```
306   \settowidth{\SR@scratchlength}{\@nameuse\SR@fontsize
307                                     \SRverbfont\selectfont~}%
```

The following statements control the line size:

```
308   \ifthenelse{\equal{\SR@linesize}{\@empty}}
309     {\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.

```
310 \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.<sup>1</sup>

```
311 \setlength{\SR@scratchlength}{\SR@verbwidth+2\tabcolsep+
312                               2\arrayrulewidth}%
```

With `\SR@verbwidth` calculated, `\SR@insert` is called to insert the `\Listing`.

```
313 \SR@insert{#2}%
314 }
```

### 3.5.2 The `\Graphic` Macro

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

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

---

<sup>1</sup>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).

321 }

`\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`.

### 3.5.3 The `\Boxlisting` Macro

The `\Boxlisting` macro creates a box that contains the verbatim content of an external listing file.

```
322 \newcommand*{\Boxlisting}[1]{%
323   \global\def\SR@insertname{\SRlistingdir/#1.lst}%
324   \IfFileExists{\SR@insertname}%
325     {%
326       \savebox{\SR@filebox}{%
327         \def\verbatim@font{\@nameuse{\SR@fontsize}\SRverbfont}%
328         \parbox{\SR@verbwidth}{%
329           \vspace*{0.25\baselineskip}%
330           \noindent\verbatiminput{\SR@insertname}%
331           \vspace*{0.25\baselineskip}}
332       }%
333     }%
334     {\global\def\SR@insertname{@empty}%
335      \savebox{\SR@filebox}{\fbox{Missing File}}}%
336   }%
337   \global\setbox\SR@filebox=\box\SR@filebox%
338   \PackageInfo{statrep}{Processing listing \SR@insertname}
339 }
```

`\Boxlisting` The `\Boxlisting` macro takes a single argument, the `{\filename}` of the listing 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 3.5.1. White space at the top and bottom of the box is inserted (`0.25\baselineskip`).

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.

### 3.5.4 The `\Boxgraphic` Macro

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

```

340 \newcommand*{\Boxgraphic}[1]{%
341   \global\def\S@insertname{\SRgraphicdir/#1.png}
342   \IfFileExists{\SR@insertname}
343     {%
344       \ifthenelse{\equal{\SR@scale}{\@empty}}
345         {\ifthenelse{\equal{\SR@width}{\@empty}}
346           {\savebox{\SR@filebox}{\includegraphics{\SR@insertname}}}
347           {\savebox{\SR@filebox}{\includegraphics[width=\SR@width]
348                                 {\SR@insertname}}}}
349       {\savebox{\SR@filebox}{\includegraphics[scale=\SR@scale]
350                                 {\SR@insertname}}}
351   }
352   {\global\def\S@insertname{\@empty}
353     \savebox{\SR@filebox}{\fbox{ Missing File }}
354   }%
355   \global\setbox\S@filebox=\box\S@filebox%
356   \PackageInfo{statrep}{Processing graphic \SR@insertname}
357   \setlength{\SR@scratchlength}{\wd\S@filebox+2\tabcolsep}%
358 }
```

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

The macro first checks whether the file exists. If the file exists, the macro includes the file. If the `width=` or `scale=` option was specified, the option is passed to the `\includegraphics` macro from the `graphicx` package.

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

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.



### 3.5.5 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, the longfigure is framed (the headings and footers include an \hline command).

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

The following statements define three macros for typing shortcuts:

```
360 \def\SR@@caption{\caption{\SRcaptionfont\SR@caption}}
361 \def\SR@@captioncontinued{\caption{\SRcaptioncontinuedfont\SRcontinuedname}}
362 \def\SR@@captionskip{\setlength{\captionmargin}{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:

```
363 \ifthenelse{\equal{\SR@type}{listing}}
364           {\def\SR@@firsthead{\hline\endLFfirsthead}
365            \def\SR@@conthead{\hline\endLFhead}
366            \def\SR@@endfoot{\hline\endLFfoot}}
367           %
368           {\def\SR@@firsthead{\endLFfirsthead}
369            \def\SR@@conthead{\endLFhead}
370            \def\SR@@endfoot{\endLFfoot}}
371 %%%
```

If the \SR@type 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:

```
372 %%%
```

```

373 \renewcommand*{\SR@firsthead}{\SR@@firsthead}%
374 \renewcommand*{\SR@conthead}{\SR@@conthead}%
375 \renewcommand*{\SR@endfoot}{\SR@@endfoot}%

376 \ifthenelse{\equal{\SR@caption}{\@empty}}%
377   {% CAPTION NOT GIVEN
378     \renewcommand*{\SR@firsthead}{}%
379     \ifSR@continued%
380       \addtocounter{\LFcounter}{-1}
381       \renewcommand*{\SR@conthead}{\SR@@captioncontinued%
382                                     \SR@@captionskip\SR@@conthead}
383     \fi%
384   }%

```

If a caption has not been specified, the first heading is empty. If the longfigure is a continuation (`\SR@continued` is true), the continuation 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.

```

385   {% CAPTION GIVEN
386     \renewcommand*{\SR@firsthead}{\SR@@caption\label{\SR@label}%
387                                     \SR@@captionskip\SR@@firsthead}%
388     \renewcommand*{\SR@conthead}{\SR@@captioncontinued%
389                                     \SR@@captionskip\SR@@conthead}
390   }
391 }

```

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` or `\Boxgraphic` macros. The length parameter `\SR@scratchlength` contains the output width.

```

392 \def\S@set@outmargin{%
393   \ifdim\S@scratchlength>\textwidth%
394     \dosloppy%
395     \setlength{\LFleft}{(\paperwidth-\SR@scratchlength)/\real{2.0}}%
396     \addtolength{\LFleft}{-1in}%
397     \addtolength{\LFleft}{-\hoffset}%
398     \addtolength{\LFleft}{-\leftskip}%
399     \ifodd\c@page\addtolength{\LFleft}{-\oddsidemargin}%
400     \else\addtolength{\LFleft}{-\evensidemargin}%

```

```

401     \fi
402     \else
403         \setlength{\Lleft}{(\textwidth-\SR@scratchlength)/\real{2.0}}%
404     \fi
405 }

```

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

$$\text{Lleft} = \frac{\text{textwidth} - \text{outputwidth}}{2}$$

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  $\text{\TeX}$  margin (1 inch)
- $\text{\hoffset}$
- $\text{\leftskip}$
- 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 +  $\text{\hoffset}$  +  $\text{\leftskip}$  + *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:

$$\text{Lleft} = -(\text{TeXmargin} + \text{hoffset} + \text{leftskip} + \text{margin}) + \frac{\text{paperwidth} - \text{outputwidth}}{2}$$

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  $\text{\hoffset}$  and  $\text{\leftskip}$  have their default values of zero width, the left margin is set as follows:

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

The  $\text{\Lleft}$  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:

```
406 \newcommand{\SR@insert}[1]{%
407   \SR@set@outheadings%
408   \ifthenelse{\equal{\SR@type}{listing}}{\Boxlisting{#1}}{\Boxgraphic{#1}}
409   \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.

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:

```
410   \ifthenelse{\equal{\SR@insertname}{\@empty}}{%
411     {\setboolean{SR@multifile}{false}}%
412     {\setboolean{SR@multifile}{true}}%
```

The following statements center a listing or a graphic and put a vertical line on each side of a listing:

```
413   \ifthenelse{\equal{\SR@type}{listing}}{
414     {\longfigure{|c|}}{\longfigure{c}}
415     \SR@firsthead\SR@conthead\SR@endfoot%
416     \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:

```
417     \setcounter{SR@multifilecount}{1}%
418     \whiledo{\boolean{SR@multifile}}{%
419       \ifthenelse{\equal{\SR@type}{listing}}{
420         {\Boxlisting{#1\arabic{SR@multifilecount}}}}%
```

```

421         {\Boxgraphic{#1\arabic{SR@multifilecount}}}%
422         \ifthenelse{\equal{\SR@insertname}{\@empty}}%
423         {\setboolean{SR@multifile}{false}}%
424         {\[0.5\baselineskip]\usebox{\SR@filebox}%
425         \stepcounter{SR@multifilecount}%
426         \setboolean{SR@multifile}{true}}%
427     }%
428     \endlongfigure%
429     \unsloppy%
430 }

```

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 differs slightly from the well-known `longtable` package written by David Carlisle. The `longtable` package defines a `longtable` environment, which produces tables that can be broken by  $\text{T}_{\text{E}}\text{X}$ 's standard page-breaking algorithm. Similarly, the `longfigure` package defines a `longfigure` environment that produces figures that can be broken by  $\text{T}_{\text{E}}\text{X}$ 's standard page-breaking algorithm.

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

- The counters and macros that start with `\LT` are renamed to start with `\LF` to avoid namespace conflicts when the two packages are used together. The generic macros that are defined in the `longtable` package (`\endfirsthead`, `\endhead`, `\endfoot`, `\endlastfoot`) are also renamed with `\LF` as a prefix.
- The `longfigure` package supports two additional key-value options:
  - `figname=` specifies the counter for numbering `longfigure` environments. The default is `figure`, but you can specify any string. If the counter is not already defined, it is created.
  - `resetby=` specifies a counter (for example, `resetby=chapter`) such that output numbering is reset with each change in the counter value. Refer to the `tocloft` package documentation for information about how the lists are typeset.

If a counter is specified that does not exist, the `tocloft` package is loaded to create the new counter.

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

```
\listoffigures
```

The default counter that is used to display figures is the `figure` counter. However, you can specify a different counter. For example, if you want to label your figures

as “Display”, specify `figname=display` when you load the `longfigure` package; to display the *List of Displays*, insert the following command in your document at the point where you want the list to appear:

```
\listofdisplay
```

When you specify a `figname` for which no counter exists, the `longfigure` package loads the `tocloft` package and creates the counter. If you want to use more advanced features of the `tocloft` package, load it before loading `longfigure`. Then the `longfigure` package sees that the counter you specified in the `figname=` option is already defined and does not attempt to create it.

**Note:** An auxiliary file with extension `.lft` is created to contain the information needed to create the list.

The `\fnum@table` macro from the `longtable` package is replaced with the `\LF@name` macro. It returns the capitalized counter name with the value of the counter. 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”.

## 5 longfigure Implementation

This section describes the implementation of the `longfigure` package. The comments describe only the changes from the `longtable` package code. For complete details about the logic and usage of the environment, see the `longtable` documentation.

```
431 \ProvidesPackage{longfigure}[2012/03/15 Multi-page Figure Package]
```

The following statement loads the `xkeyval` package to declare and process package options.

```
432 \RequirePackage{xkeyval}
```

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

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

The following statement defines a new command, `\LFreset`, to contain the name of the counter that `longfigure` number should reset within. If no value is given, the `longfigures` are numbered consecutively through the document.

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

## 5.1 Options

The two commands that were just defined exist to support the package options `figname=` and `resetby=`:

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

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

```
437 \DeclareOptionX{set}{}
438 \DeclareOptionX{final}{}
439 \DeclareOptionX{errorshow}{\def\LF@warn{\PackageInfo{longfigure}}}
440 \DeclareOptionX{pausing}{\def\LF@warn#1{\LF@err{#1}{This is not really an error}}}
441 \ProcessOptionsX
```

The following statements process the options:

```
442 \def\LFProcessOptions#1{
443   \@ifundefined{c@#1}{%
444     \RequirePackage{tocloft}
445     \expandafter\def\csname list#1name\endcsname{List of #1s}
446     \ifx\@empty\LFreset%
447       \newlistof{#1}{lft}{\csname list#1name\endcsname}
448     \else
449       \newlistof[\LFreset]{#1}{lft}{\csname list#1name\endcsname}
450     \fi
451   \fi
452 }{}%
453 }
454 \expandafter\LFProcessOptions\expandafter{\LFcounter}
```

If a counter is specified that does not exist, `\c@countername` is undefined and the `tocloft` package is loaded in order to use its commands to create the new counters and list.

Thus, the extra package is required only when a new counter is specified. 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 `longfigure` and retain all the flexibility that package offers. However, you must



define the new counters by using the `tocloft` package command `\newlistof`. You must also define the new list to use the `lft` file for writing its auxiliary information.

## 5.2 Utilities

`\strcfstr` The following macro, `\strcfstr`, is described by Wilson (2001). The purpose of the macro is to check whether two strings, specified as arguments, are equal. A new Boolean switch, `\ifLF@same`, contains the result of the test.

```
455 \newif\ifLF@same
456 \newcommand{\strcfstr}[2]{%
457   \LF@samefalse
458   \begingroup\def\2{#2}
459   \ifx\2#1\endgroup\LF@sametrue
460   \else\endgroup
461   \fi
462 }
```

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

`\LFupcase`

```
463 \def\LFupcase#1{%
464   \def\x##1##2{%
465     \MakeUppercase{##1}{##2}}\x#1%
466 }
```

The following macro creates a string to provide a label and number for an output. It replaces `\fnum@table` from the `longtable` package. It contains the capitalized version of the counter name and the counter number (for example, Figure~3).

```
467 \def\LF@name{\expandafter\LFupcase%
468             \expandafter{\LFcounter}~%
469             \expandafter\csname the\LFcounter\endcsname}%

```

The remainder of this package follows the `longtable` package almost identically, except that macros, skips, counters, and so on are prefixed with `\LF` instead of `\LT` as in the `longtable` package:

```
470 \def\LF@err{\PackageError{longfigure}}
471 \def\LF@warn{\PackageWarning{longfigure}}
472 \def\LF@final@warn{%
473   \AtEndDocument{%

```

```

474 \LF@warn{\LFcounter \@width s have changed. Rerun \LaTeX\.\@gobbletwo}}%
475 \global\let\LF@final@warn\relax}
476 %
477 \newskip\LFleft \LFleft=\fill
478 \newskip\LFright \LFright=\fill
479 \newskip\LFpre \LFpre=\bigskipamount
480 \newskip\LFpost \LFpost=\bigskipamount
481 \newcount\LFchunksize \LFchunksize=20
482 \let\c@LFchunksize\LFchunksize
483 \newdimen\LFcapwidth \LFcapwidth=4in
484 \newbox\LF@head
485 \newbox\LF@firsthead
486 \newbox\LF@foot
487 \newbox\LF@lastfoot
488 \newcount\LF@cols
489 \newcount\LF@rows
490 \newcounter{LF@tables}
491 \newcounter{LF@chunks}[LF@tables]
492 %
493 \newtoks\LF@p@ftn
494 \mathchardef\LF@end@pen=30000
495 \def\longfigure{%
496 \par
497 \ifx\multicols\@undefined
498 \else
499 \ifnum\col@number>\@ne
500 \@twocolumntrue
501 \fi
502 \fi
503 \if@twocolumn
504 \LF@err{longfigure not in 1-column mode}\@ehc
505 \fi
506 \begingroup
507 \@ifnextchar[\LF@array{\LF@array[x]}}
508 \def\LF@array[#1]#2{%
509 \refstepcounter{\LFcounter}\stepcounter{LF@tables}%
510 \if l#1%
511 \LFleft\z@ \LFright\fill
512 \else\if r#1%
513 \LFleft\fill \LFright\z@
514 \else\if c#1%
515 \LFleft\fill \LFright\fill
516 \fi\fi\fi
517 \let\LF@mcol\multicolumn
518 \let\LF@@tabarray\@tabarray

```

```

519 \let\LF@@hl\hline
520 \def\@tabarray{%
521   \let\hline\LF@@hl
522   \LF@@tabarray}%
523 \let\\\LF@tabularcr\let\tabularnewline\\%
524 \def\newpage{\noalign{\break}}%
525 \def\pagebreak{\noalign{\ifnum' }=0\fi\@testopt{\LF@no@pgbk-}4}%
526 \def\nopagebreak{\noalign{\ifnum' }=0\fi\@testopt{\LF@no@pgbk4}}%
527 \let\hline\LF@hline \let\kill\LF@kill\let\caption\LF@caption
528 \@tempdima\ht\strutbox
529 \let\@endpbox\LF@endpbox
530 \ifx\extrarowheight\@undefined
531   \let\@acol\@tabacol
532   \let\@classz\@tabclassz \let\@classiv\@tabclassiv
533   \def\@startpbox{\vtop\LF@startpbox}%
534   \let\@@startpbox\@startpbox
535   \let\@@endpbox\@endpbox
536   \let\LF@LL@FM@cr\@tabularcr
537 \else
538   \advance\@tempdima\extrarowheight
539   \col@sep\tabcolsep
540   \let\@startpbox\LF@startpbox\let\LF@LL@FM@cr\@arraycr
541 \fi
542 \setbox\@arstrutbox\hbox{\vrule
543   \@height \arraystretch \@tempdima
544   \@depth \arraystretch \dp \strutbox
545   \@width \z@}%
546 \let\@sharp##\let\protect\relax
547 \begingroup
548   \@mkpream{#2}%
549   \xdef\LF@bchunk{%
550     \global\advance\c@LF@chunks\@ne
551     \global\LF@rows\z@\setbox\z@\vbox\bgroup
552     \LF@setprevdepth
553     \tabskip\LF@left \noexpand\halign to\hsize\bgroup
554     \tabskip\z@ \@arstrut \@preamble \tabskip\LF@right \cr}%
555 \endgroup
556 \expandafter\LF@nofcols\LF@bchunk&\LF@nofcols
557 \LF@make@row
558 \m@th\let\par\@empty
559 \everycr{}\lineskip\z@\baselineskip\z@
560 \LF@bchunk}
561 \def\LF@no@pgbk#1[#2]{\penalty #1\@getpen{#2}\ifnum'={0\fi}}
562 \def\LF@start{%
563   \let\LF@start\endgraf

```

```

564 \endgraf\penalty\z@\vskip\LFpre
565 \dimen@ \pagetotal
566 \advance\dimen@ \ht\ifvoid\LF@firsthead\LF@head\else\LF@firsthead\fi
567 \advance\dimen@ \dp\ifvoid\LF@firsthead\LF@head\else\LF@firsthead\fi
568 \advance\dimen@ \ht\LF@foot
569 \dimen@ii\vfuzz
570 \vfuzz\maxdimen
571 \setbox\tw@\copy\z@
572 \setbox\tw@\vsplit\tw@ to \ht\@arstrutbox
573 \setbox\tw@\vbox{\unvbox\tw@}%
574 \vfuzz\dimen@ii
575 \advance\dimen@ \ht
576 \ifdim\ht\@arstrutbox>\ht\tw@\@arstrutbox\else\tw@\fi
577 \advance\dimen@\dp
578 \ifdim\dp\@arstrutbox>\dp\tw@\@arstrutbox\else\tw@\fi
579 \advance\dimen@ -\pagegoal
580 \ifdim \dimen@>\z@\vfil\break\fi
581 \global\@colroom\@colht
582 \ifvoid\LF@foot\else
583 \advance\vsizel-\ht\LF@foot
584 \global\advance\@colroom-\ht\LF@foot
585 \dimen@\pagegoal\advance\dimen@-\ht\LF@foot\pagegoal\dimen@
586 \maxdepth\z@
587 \fi
588 \ifvoid\LF@firsthead\copy\LF@head\else\box\LF@firsthead\fi\nobreak
589 \output{\LF@output}}
590 \def\endlongfigure{%
591 \crr
592 \noalign{%
593 \let\LF@entry\LF@entry@chop
594 \xdef\LF@save@row{\LF@save@row}}%
595 \LF@echunk
596 \LF@start
597 \unvbox\z@
598 \LF@get@widths
599 \if@filesw
600 {\let\LF@entry\LF@entry@write\immediate\write\@auxout{%
601 \gdef\expandafter\noexpand
602 \csname LF@\romannumeral\c@LF@tables\endcsname
603 {\LF@save@row}}}%
604 \fi
605 \ifx\LF@save@row\LF@@save@row
606 \else
607 \LF@warn{Column \@width s have changed\MessageBreak
608 in table \thetable}%

```

```

609 \LF@final@warn
610 \fi
611 \endgraf\penalty -\LF@end@pen
612 \endgroup
613 \global\@mparbottom\z@
614 \pagegoal\vsizer
615 \endgraf\penalty\z@\addvspace\LFpost
616 \ifvoid\footins\else\insert\footins{}\fi}
617 \def\LF@nofcols#1&{%
618 \futurelet\@let@token\LF@n@fcols}
619 \def\LF@n@fcols{%
620 \advance\LF@cols\@ne
621 \ifx\@let@token\LF@nofcols
622 \expandafter\@gobble
623 \else
624 \expandafter\LF@nofcols
625 \fi}
626 \def\LF@tabularcr{%
627 \relax\iffalse{\fi\ifnum0='}\fi
628 \@ifstar
629 {\def\crrcr{\LF@crrcr\noalign{\nobreak}}\let\cr\crrcr
630 \LF@t@bularcr}%
631 {\LF@t@bularcr}}
632 \let\LF@crrcr\crrcr
633 \let\LF@setprevdepth\relax
634 \def\LF@t@bularcr{%
635 \global\advance\LF@rows\@ne
636 \ifnum\LF@rows=\LFchunksize
637 \gdef\LF@setprevdepth{%
638 \prevdepth\z@\global
639 \global\let\LF@setprevdepth\relax}%
640 \expandafter\LF@xtabularcr
641 \else
642 \ifnum0='{}\fi
643 \expandafter\LF@LL@FM@cr
644 \fi}
645 \def\LF@xtabularcr{%
646 \@ifnextchar[\LF@argtabularcr\LF@ntabularcr}
647 \def\LF@ntabularcr{%
648 \ifnum0='{}\fi
649 \LF@echunk
650 \LF@start
651 \unvbox\z@
652 \LF@get@widths
653 \LF@bchunk}

```

```

654 \def\LF@argtabularcr[#1]{%
655   \ifnum0='{ }\fi
656   \ifdim #1>\z@
657     \unskip\@xargarraycr{#1}%
658   \else
659     \@yargarraycr{#1}%
660   \fi
661   \LF@echunk
662   \LF@start
663   \unvbox\z@
664   \LF@get@widths
665   \LF@bchunk}
666 \def\LF@echunk{%
667   \crcr\LF@save@row\cr\egroup
668   \global\setbox\@ne\lastbox
669   \unskip
670   \egroup}
671 \def\LF@entry#1#2{%
672   \ifhmode\@firstofone{&}\fi\omit
673   \ifnum#1=\c@LF@chunks
674   \else
675     \kern#2\relax
676   \fi}
677 \def\LF@entry@chop#1#2{%
678   \noexpand\LF@entry
679   {\ifnum#1>\c@LF@chunks
680     1}{0pt%
681     \else
682       #1}{#2%
683     \fi}}
684 \def\LF@entry@write{%
685   \noexpand\LF@entry^^J%
686   \@spaces}
687 \def\LF@kill{%
688   \LF@echunk
689   \LF@get@widths
690   \expandafter\LF@rebox\LF@bchunk}
691 \def\LF@rebox#1\bgroup{%
692   #1\bgroup
693   \unvbox\z@
694   \unskip
695   \setbox\z@\lastbox}
696 \def\LF@blank@row{%
697   \xdef\LF@save@row{\expandafter\LF@build@blank
698     \romannumeral\number\LF@cols 001 }}

```

```

699 \def\LF@build@blank#1{%
700   \if#1m%
701     \noexpand\LF@entry{1}{0pt}%
702     \expandafter\LF@build@blank
703   \fi}
704 \def\LF@make@row{%
705   \global\expandafter\let\expandafter\LF@save@row
706     \csname LF@\romannumeral\c@LF@tables\endcsname
707   \ifx\LF@save@row\relax
708     \LF@blank@row
709   \else
710     {\let\LF@entry\or
711      \if!%
712        \ifcase\expandafter\expandafter\expandafter\LF@cols
713          \expandafter\@gobble\LF@save@row
714        \or
715        \else
716          \relax
717        \fi
718      !%
719      \else
720        \aftergroup\LF@blank@row
721      \fi}%
722   \fi}
723 \let\setlongfigures\relax
724 \def\LF@get@widths{%
725   \setbox\tw@\hbox{%
726     \unhbox\@ne
727     \let\LF@old@row\LF@save@row
728     \global\let\LF@save@row\@empty
729     \count@\LF@cols
730     \loop
731       \unskip
732       \setbox\tw@\lastbox
733       \ifhbox\tw@
734         \LF@def@row
735         \advance\count@\m@ne
736       \repeat}%
737   \ifx\LF@@save@row\@undefined
738     \let\LF@@save@row\LF@save@row
739   \fi}
740 \def\LF@def@row{%
741   \let\LF@entry\or
742   \edef\@tempa{%
743     \ifcase\expandafter\count@\LF@old@row

```

```

744     \else
745         {1}{0pt}%
746     \fi}%
747 \let\LF@entry\relax
748 \xdef\LF@save@row{%
749     \LF@entry
750     \expandafter\LF@max@sel\@tempa
751     \LF@save@row}}
752 \def\LF@max@sel#1#2{%
753     {\ifdim#2=\wd\tw@
754         #1%
755     \else
756         \number\c@LF@chunks
757     \fi}%
758     {\the\wd\tw@}}
759 \def\LF@hline{%
760     \noalign{\ifnum0='}\fi
761     \penalty\@M
762     \futurelet\@let@token\LF@@hline}
763 \def\LF@@hline{%
764     \ifx\@let@token\hline
765         \global\let\@gtempa\@gobble
766         \gdef\LF@sep{\penalty-\@medpenalty\vskip\doublerulesep}%
767     \else
768         \global\let\@gtempa\@empty
769         \gdef\LF@sep{\penalty-\@lowpenalty\vskip-\arrayrulewidth}%
770     \fi
771     \ifnum0='\fi}%
772 \multispan\LF@cols
773     \unskip\leaders\hrule\@height\arrayrulewidth\hfill\cr
774 \noalign{\LF@sep}%
775 \multispan\LF@cols
776     \unskip\leaders\hrule\@height\arrayrulewidth\hfill\cr
777 \noalign{\penalty\@M}%
778 \@gtempa}

```

### 5.3 Captioning

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

```

779 \def\LF@caption{%

```



```

780 \noalign\bgroup
781   \@ifnextchar[{\egroup\LF@c@ption\@firstofone}\LF@capti@n}

```

\LF@c@ption begins the process. If it has an optional argument, it calls \LF@c@ption; otherwise it calls \LF@capti@n, which then calls \LF@c@ption.

```

782 \def\LF@c@ption#1[#2]#3{%
783   \LF@makecaption#1\LF@name{#3}%
784   \def\@tempa{#2}%
785   \ifx\@tempa\@empty\else

```

If a list of longfigures is requested:

- 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.

The previously defined macro \strcfstr and Boolean switch \ifLF@same are used here to determine the name of the counter and set the output file to contain the longfigure information:

```

786   {\let\\\space
787   \strcfstr{\LFcounter}{figure}
788   \ifLF@same\def\LFoutfile{lof}\else
789   \strcfstr{\LFcounter}{table}
790   \ifLF@same\def\LFoutfile{lot}\else
791   \def\LFoutfile{lft}\fi\fi
792   \addcontentsline{\LFoutfile}{\LFcounter}
793   {\expandafter\protect\expandafter\numberline\expandafter%
794   {\expandafter\csname the\LFcounter\endcsname}{#2}}}%
795   \fi
796 }

```

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

```

797 \def\LF@capti@n{%
798   \@ifstar
799   {\egroup\LF@c@ption\@gobble[]}%
800   {\egroup\@xdblarg{\LF@c@ption\@firstofone}}}

```

To redefine how the longfigure is captioned, use 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.

```

801 \def\LF@makecaption#1#2#3{%
802   \LF@mcol\LF@cols c{\hbox to\z@{\hss\parbox[t]\LF@capwidth{%
803     \sbox\@tempboxa{#1{#2: }#3}%
804     \ifdim\wd\@tempboxa>\hsize
805       #1{#2: }#3%
806     \else
807       \hbox to\hsize{\hfil\box\@tempboxa\hfil}%
808     \fi
809     \endgraf\vskip\baselineskip}%
810   \hss}}
811 \def\LF@output{%
812   \ifnum\outputpenalty <-\@Mi
813     \ifnum\outputpenalty > -\LF@end@pen
814       \LF@err{floats and marginpars not allowed in a longfigure}\@ehc
815     \else
816       \setbox\z@\vbox{\unvbox\@cclv}%
817       \ifdim \ht\LF@lastfoot>\ht\LF@foot
818         \dimen@\pagegoal
819         \advance\dimen@-\ht\LF@lastfoot
820         \ifdim\dimen@<\ht\z@
821           \setbox\@cclv\vbox{\unvbox\z@\copy\LF@foot\vss}%
822           \@makecol
823           \@outputpage
824           \setbox\z@\vbox{\box\LF@head}%
825         \fi
826       \fi
827       \global\@colroom\@colht
828       \global\vsizel\@colht
829       \vbox
830       {\unvbox\z@\box\ifvoid\LF@lastfoot\LF@foot\else
831        \LF@lastfoot\fi}%
832     \fi
833   \else
834     \setbox\@cclv\vbox{\unvbox\@cclv\copy\LF@foot\vss}%
835     \@makecol
836     \@outputpage
837     \global\vsizel\@colroom
838     \copy\LF@head\nobreak
839   \fi}
840 \def\LF@end@hd@ft#1{%
841   \LF@echunk
842   \ifx\LF@start\endgraf
843     \LF@err
844     {Longfigure head or foot not at start of table}%
845     {Increase LFchunksizel}%

```

```

846 \fi
847 \setbox#1\box\z@
848 \LF@get@widths
849 \LF@bchunk}

```

The following four macros (`\endLFfirsthead`, `\endLFhead`, `\endLFfoot`, and `\endLFlastfoot`) are also defined to have a prefix of `\LF`. They do not have an `\LT` prefix in the `longtable` package, but they must be redefined to avoid a namespace clash.

```

850 \def\endLFfirsthead{\LF@end@hd@ft\LF@firsthead}
851 \def\endLFhead{\LF@end@hd@ft\LF@head}
852 \def\endLFfoot{\LF@end@hd@ft\LF@foot}
853 \def\endLFlastfoot{\LF@end@hd@ft\LF@lastfoot}
854 %
855 \def\LF@startpbox#1{%
856   \bgroup
857   \let\@footnotetext\LF@p@ftntext
858   \setlength\hsize{#1}%
859   \@arrayparboxrestore
860   \vrule \@height \ht\@arstrutbox \@width \z@}
861 \def\LF@endpbox{%
862   \@finalstrut\@arstrutbox
863   \egroup
864   \the\LF@p@ftn
865   \global\LF@p@ftn{}}%
866 \hfil}
867 \def\LF@p@ftntext#1{%
868   \edef\@tempa{\the\LF@p@ftn\noexpand\footnotetext[\the\c@footnote]}}%
869   \global\LF@p@ftn\expandafter{\@tempa{#1}}}%

```

## 5.4 References

Carlisle, D. 2004. *The longtable Package*. Included in the “Comprehensive T<sub>E</sub>X Archive Network.” <http://ctan.org>.

Lazarides, Y. 2010. T<sub>E</sub>Xstackexchange, online forum. <http://tex.stackexchange.com/questions/7992>.

Schöpf, R, B. Raichle, and C. Rowley. 2001. *A New Implementation of L<sup>A</sup>T<sub>E</sub>X’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](http://www.tug.org/applications/pdftex).

Wilson, P. 2001. *Glistings*. 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.

Symbols			
\# . . . . . 30, 277	\AtBeginDocument 121	<b>E</b>	
\% 187, 188, 191, 199, 205, 210, 211, 216	\AtEndDocument . . 122	\endverbatim . . . . 263	
\* . . . 193, 195, 201, 207	<b>B</b>	environments:	
\@Datastep . . 137, 143	\Boxgraphic . . . . .	Datastep . . . . . 145	
\@Sascode . . . 244, 261	. . . . . 340, 359, 408	Sascode . . . . . 244	
\@backslashchar . .	\Boxlisting . . . . .	\evensidemargin . 400	
. . . . . 63, 66, 104	. . . . . 322, 340, 408	<b>F</b>	
\@bsphack . . . . . 125	<b>C</b>	\fbox . . . . . 335, 353	
\@esphack . . . . . 134	\caption . . . . . 360, 361	figname option . . . . . 3	
\@gobbletwo . . . . . 13	caption option . . . . . 6	first option . . . . . 6	
\@nameuse . . . . . 7, 151, 254, 306, 327	\catcode . . . . . 30, 126, 142, 143, 187, 188, 191, 199, 205, 210, 211, 216, 260, 261	firstobj option . . . . . 7	
\@percentchar . . . .	category codes . . . . . 19	fontsize option . . . . . 6	
. . . . . 59, 61, 62, 64, 65, 95, 96, 97, 98, 99, 100, 113, 249, 271, 279	\closeout 110, 122, 134	<b>G</b>	
\@tempwafalse 199, 205, 210, 223, 231	continued option . . . . . 6	generate option . . . . . 3	
# . . . . . 27	\CurrentOption . . . 8, 9	\Graphic . . . . . 315, 322	
\^ 126, 142, 143, 260, 261	customizing . . . . . 8	<b>H</b>	
<b>A</b>	<b>D</b>	hashchar . . . . . 27	
\active . . 126, 142, 260	Datastep (environ- ment) . . . . . 145	height option . . . . . 7	
\addtocounter 129, 174, 230, 235, 380	Datastep processing . . 18	\hfuzz . . . . . 15, 16	
\addtolength . 396, 397, 398, 399, 400	Datastep schematic . . 14	<b>I</b>	
	display option . . . . . 6	\IfFileExists 324, 342	
	\dosloppy . . . 5, 15, 394	\ifodd . . . . . 399	
	dpi option . . . . . 7	\ifSR@continued . 379	
		\ifSR@display 155, 182, 225, 246, 268	
		\ifSR@generate . 6, 12	

\ifSR@program . . . .	<b>S</b>	\SR@hashchar 6, 30, 277
147, 178, 221, 266	Sascode (environ-	\SR@insert 313, 320, 406
\includegraphics .	ment) . . . . . 244	\SR@insertname . . .
. . . . 346, 347, 349	Sascode processing . .	. 323, 324, 330,
\InputIfFileExists 75	. . . . . 21, 24, 25	334, 338, 341,
<b>J</b>	Sascode schematic . . . 24	342, 346, 348,
\jobname . . . . . 57, 58	scale option . . . . . 6	350, 352, 356, 410
<b>L</b>	\setkeys . . . . .	\SR@label . . . 279, 386
last option . . . . . 6	138, 245, 299, 316	\SR@last . . . . 153, 163
lastobj option . . . . . 7	\SR@_keys . . . . . 6	\SR@linesize . . . .
\LFleft . . . . .	\SR@@caption . 360, 386	. . . . 308, 309, 310
. 395, 396, 397,	\SR@@captioncontinued	\SR@outkeyval . . . .
398, 399, 400, 403	. . . . 361, 381, 388	. . . . . 28, 285,
linesize option . . . . . 7	\SR@@captionskip .	286, 287, 288, 290
list of outputs, creating . 3	362, 382, 387, 389	\SR@preambletext .
\Listing . . . . . 298	\SR@@conthead 365,	. . . . . 81, 93, 112
\longfigure . . . . 414	369, 374, 382, 389	\SR@program@stream
<b>N</b>	\SR@@endfoot . . . . .	. 79, 111, 112,
\newboolean . . . . . 29	. . . . 366, 370, 375	113, 114, 115,
<b>O</b>	\SR@@firsthead . . .	116, 117, 118,
objects option . . . . . 7	364, 368, 373, 387	119, 122, 136,
\oddsidemargin . . 399	\SR@caption . 360, 376	183, 209, 222,
\openout . 92, 111, 125	\SR@conthead . . 21,	229, 237, 249,
output centering . . . . 34	374, 381, 388, 415	271, 272, 278, 281
output labels . . . . . 29	\SR@datastep@writefile	\SR@sascode@addtoline
output processing . . . 34	. . . . 14, 123, 139	. . . . 19, 188, 257
<b>P</b>	\SR@datastep@writeprogram	\SR@sascode@filter
\PackageInfo . . 76,	. 14, 135, 147, 155	. . . . 20, 190, 192
77, 250, 338, 356	\SR@enddatastep@writefile	\SR@sascode@writeline
pagesize option . . . . . 7	. . . . 14, 134, 146	. 22, 211, 216, 220
\paperwidth . . . . 395	\SR@endfoot 22, 375, 415	\SR@scale . . . 344, 349
\PassOptionsToPackage	\SR@filebox 19, 326,	\SR@scratchlength
. . . . . 8, 9	335, 337, 346,	. . . . . 17, 179,
pattern option . . . . . 7	347, 349, 353,	180, 264, 265,
\pdflastmatch . . . .	355, 357, 359, 416	306, 310, 311,
. 20, 198, 204, 209	\SR@final@outoptions	357, 393, 395, 403
\pdfmatch . . . . 20,	. . . . 279, 284, 295	\SR@set@outheadings
193, 195, 201, 207	\SR@first 156, 157, 160	. . . . 33, 359, 407
png graphics . . . . . 32	\SR@firsthead . 20,	\SR@set@outmargin
\presetkeys . . . 8, 36	373, 378, 386, 415	. . . . 34, 392, 409
program option . . . . . 6	\SR@fontsize . 149,	\SR@set@outoptions
	151, 252, 254,	. . . . 28, 276, 283
	303, 304, 306, 327	\SR@store . . . 247,
	\SR@generate . . . . . 4	249, 250, 269,
	\SR@generatetrue . . 6	271, 279, 300, 317

\SR@tempfile@stream	..... 10, 71, 361	<b>T</b>
..... 80, 92,	\SRdpi ..... 9, 53, 97	\tempfilename .... 10
93, 94, 95, 96,	\SRgraphicdir ....	\textwidth .. 393, 403
97, 98, 99, 100,	... 9, 55, 100, 341	\theSR@currentline
101, 102, 103,	\SRintertext 10, 69, 161	157, 160, 164, 165
104, 105, 106,	\SRlinesize .....	\theSR@displayline
107, 108, 109,	.... 9, 51, 95, 309	..... 233
110, 125, 130, 134	\SRlistingdir ....	\theSR@programline
\SR@type . 363, 408, 413	.... 9, 56, 99, 323	..... 228
\SR@verwidth ....	\SRmacroinclude ..	\theSR@startinglastline
. 18, 310, 311, 328	..... 10, 59, 108	..... 164, 165
\SR@width ... 345, 347	\SRmacropath . 8, 49, 59	\theSR@totallines 153
\SR@write@outoptions	\SRpagesize . 9, 52, 96	type option ..... 7
. 27, 275, 302, 319	\SRparindent 10, 68,	<b>U</b>
\SR@write@preamble	158, 161, 166,	\unsloppy ..... 5, 16
..... 12, 91, 121	172, 226, 234, 238	<b>V</b>
\SR@writepgm .....	\SRpreamblename ..	\verbatim ..... 258
. 4, 12, 13, 93,	..... 58, 92, 113	verbatim environment 17
94, 95, 96, 97,	\SRprogramline ... 10	\verbatim@addtoline
98, 99, 100, 101,	\SRprogramname ...	..... 257
102, 103, 104,	..... 10, 57, 111	\verbatim@font .. 327
105, 106, 107,	\SRsasprogramline	\verbatim@line 130,
108, 109, 112,	..... 60, 115	136, 158, 166,
113, 114, 115,	\SRstyle .... 9, 54, 98	172, 199, 205,
116, 117, 118,	\SRtempfilename ..	210, 226, 234, 238
119, 130, 136,	.... 70, 139, 177	\verbatim@processline
183, 209, 222,	\SRverbfont .....	. 16, 128, 147, 154
229, 237, 249,	.. 10, 74, 307, 327	\verbatim@start . 132
271, 272, 278, 281	statrep package ..... 2	\verbatiminput ...
\SRcaptioncontinuedfont	statrep.cfg ..... 8	..... 177, 330
..... 10, 73, 361	store option ..... 7	<b>W</b>
\SRcaptionfont ...	\strip@prefix ....	width option ..... 7
..... 10, 72, 360	..... 198, 204, 209	
\SRcontinuedname .	style option ..... 7	