

The statrep package*

Martial Phélippe-Guinvarc'h

Le Mans Université

`martial.phelippe-guinvarch@univ-lemans.fr`

Based on the original work by Tim Arnold and Warren F. Kuhfeld (SAS Institute Inc. 2015)

2026/04/28

Contents

1	About This Document	2
2	StatRep Usage	2
2.1	Workflow	3
3	StatRep Implementation	4
3.1	Required Packages	4
3.2	Programming Utilities	5
3.3	Customizable Settings	10
3.4	Character Encoding (<code>SASencodingOutput</code>)	12
3.5	Code Environments	15
3.5.1	Datastep Environment	16
3.5.2	Sascode Environment	20
3.6	Handling the SAS Output	27
3.6.1	Note on Widths	29
3.6.2	The <code>\Listing</code> Macro	29
3.6.3	The <code>\Graphic</code> Macro	31
3.6.4	The <code>\Boxlisting</code> Macro	31
3.6.5	The <code>\Boxgraphic</code> Macro	33
3.6.6	The <code>\SR@insert</code> Macro	34
3.7	SAS Result Export Macros	38
4	longfigure Usage	39
4.1	Example	41
5	Implementation	42

*This document corresponds to statrep v2.0, last revised 2026/04/28.

5.1	Options	42
5.2	Utilities	43
5.3	Captioning	50
5.4	References	53

List of Figures

1	Example of Using the StatRep Package	2
2	Processing Schematic for the Datastep Environment	17
3	Processing Schematic for the Sascode Environment	25
4	Example of %STATREPALLRESULTS Usage	39

1 About This Document

This document uses the following aids for presenting information.

- The verbatim font (`example text`) is used for names of variables, macros, and environments.
- Hyperlinks are displayed in red.
- Figures are displayed with a light blue background.
- Macro names are given in the margin near where the macro is defined or described.
- The index contains terms with pointers to the code lines.

2 StatRep Usage

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

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

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

Figure 1: Example of Using the StatRep Package

The StatRep package supports the following options:

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

- `figname=` specifies the name of a \LaTeX counter that is used for numbering outputs. The default is `figure`. If you specify a value for the `figname` option for which no counter exists, a counter is created.
- `resetby=` specifies that the counter for output numbering be reset with each change in the specified counter value. For example, if `resetby=chapter`, all output numbering is reset when the chapter value changes. See section 4 for details. Also, refer to the `tocloft` package documentation for information about how the lists are typeset.
- `SASmode=WorkStation` or `SASmode=Server` changes the current path definition. For `WorkStation`, the `rootdir` or current path is `.` (the working directory); for `Server`, the current path is `¤tpath`, obtained from the `&_SASPROGRAMFILE` SAS macro variable.

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.

2.1 Workflow

To produce a complete document with StatRep, follow these steps:

1. Compile the document with `pdf \LaTeX` . You can use a \LaTeX -aware editor such as `T \TeX works`, or you can use the command-line command `pdflatex`. This step generates the necessary SAS program that is needed to produce the results. If the name of your document is *myarticle.tex*, the names of the generated SAS programs will be *myarticle_SR.sas* and *myarticle_SR_preamble.sas* by default.

myarticle_SR.sas Begins with a call to the preamble SAS file, then the

cleaning subdirectory, and then the aggregate SAS code present in the \LaTeX file.

myarticle_SR_preamble.sas Mainly contains custom variables (see section 3.3) and a call to the `statrep_macros.sas` macro file.

2. Execute the generated SAS program *myarticle_SR.sas*. This step generates the SAS results requested in your document.
3. Recompile the document with pdf \LaTeX . This step includes the SAS results in the final document. You might need to repeat this step so that \LaTeX can re-measure the listing outputs to ensure that they are framed properly.

3 StatRep Implementation

3.1 Required Packages

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

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

- 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.
- The `etoolbox` is needed for string comparison (`ifstrequal`)

```
7 \newif\ifSASServer
8 \SASServerfalse % Default is Workstation
9 % Define the 'SASmode' key for the 'statrep' package
10 \DeclareOptionX{SASmode}{\setkeys{statrep}{SASmode=#1}}
11 \define@key{statrep}{SASmode}{%
12   \ifstrequal{#1}{Server}{% if value is 'Server'
13     \SASServertrue
14     \PackageInfo{statrep}{Mode SAS: Serveur (utilise \string&\string_SASPROGRAMFILE).
15   }{% If value is not 'Serveur', we control if it is 'WorkStation'
16     \ifstrequal{#1}{WorkStation}{%
17       \SASServerfalse
18       \PackageInfo{statrep}{Mode SAS: WorkStation (use ./).}
19     }{%
```

```

20      % If value is unknown, we print an warning
21      \SASServerfalse
22      \PackageWarning{statrep}{Unknown value for SASmode (#1). Default use of WorkSta
23    }%
24  }%
25 }

```

The SASmode option is defined to provide an explicit current path when the default "/" does not work (SAS Server/SAS on Demand, etc.).

```

26 \newif\ifSR@generate\SR@generatetrue
27 \DeclareOptionX{generate}[true]{\@nameuse{SR@generate#1}}
28 \DeclareOptionX{color}[true]{%
29   \IfFileExists{sas.sty}
30   {\PassOptionsToPackage{\CurrentOption}{sas}}
31   {\relax}
32 }
33 \DeclareOptionX{figname}{\PassOptionsToPackage{\CurrentOption}{longfigure}}
34 \DeclareOptionX{resetby}{\PassOptionsToPackage{\CurrentOption}{longfigure}}
35 \ProcessOptionsX

```

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

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

3.2 Programming Utilities

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

```

36 \providecommand{\SR@encoding}{utf-8}
37 \def\statrep@map@sas@encoding#1{%
38   \def\SR@encoding{utf-8}
39   \ifstrequal{#1}{utf8}{\def\SR@encoding{utf-8}}{}%
40   \ifstrequal{#1}{utf8x}{\def\SR@encoding{utf-8}}{}%
41   % latin1, latin9, ansinew, etc. is mapped to wlatin1
42   \ifstrequal{#1}{latin1}{\def\SR@encoding{wlatin1}}{}%
43   \ifstrequal{#1}{latin9}{\def\SR@encoding{wlatin1}}{}%
44   \ifstrequal{#1}{ansinew}{\def\SR@encoding{wlatin1}}{}%
45 }
46 \AtBeginDocument{%
47   \ifpackageloaded{inputenc}{%
48     \statrep@map@sas@encoding{\inputencodingname}%
49     \PackageInfo{statrep}{Encodind LaTeX ({\inputencodingname}) detected and mapped
50     \SR@encoding.}

```

```

51    }{%
52    % else, default utf-8 is retained.
53    \PackageInfo{statrep}{inputenc unfund. Default SAS encoding: \SR@encoding utf-
54    }
55 }
56 \IfFileExists{sas.sty}
57 {\RequirePackage{sas}}
58 {\relax}
59 \IfFileExists{\jobname _VarSASToTex.tex}{\input{\jobname _VarSASToTex.tex}}{}
60 \RequirePackage{longfigure}

```

Management of the encoding of the input SAS file Control of the presence of the SAS.sty product file in the SAS compilation

```

61 \ifSR@generate\def\SR@writepgm{\immediate\write}
62 \else\let\SR@writepgm\@gobbletwo
63 \fi

```

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

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

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

```

64 \newcommand*{\dosloppy}{\setlength{\hfuzz}{\maxdimen}\hbadness\maxdimen}
65 \newcommand*{\unsloppy}{\setlength{\hfuzz}{0.3pt}\hbadness 1414}

```

`\dosloppy` The `dosloppy` macro is intended to be used when you knowingly violate typesetting 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^AT_EX'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 3.6.6 for details.

The following definitions are used to calculate and set the output stream of listings or figures. See section 3.6.6 for details.

```

66 \newlength{\SR@scratchlength}
67 \newlength{\SR@verbwidth}
68 \newsavebox{\SR@filebox}

```

```

69 \newcommand*\SR@firsthead{}\{}
70 \newcommand*\SR@conthead{}\{}
71 \newcommand*\SR@endfoot{}\{}

```

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

```

72 \newcounter{SR@currentline}\setcounter{SR@currentline}{0}
73 \newcounter{SR@displayline}\setcounter{SR@displayline}{0}
74 \newcounter{SR@programline}\setcounter{SR@programline}{0}
75 \newcounter{SR@startinglastline}\setcounter{SR@startinglastline}{0}
76 \newcounter{SR@totallines}\setcounter{SR@totallines}{0}
77 \newcounter{SR@multifilecount}\setcounter{SR@multifilecount}{0}
78 \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 3.6.6 for complete details.

```

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

```

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

The `\SR@hashchar` macro is used in the `\SR@write@outoptions` macro. See section 3.6 for details.

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

```
80 \define@boolkeys{SR}{SR@}{display,program,continued}[true]{}
81 \define@cmdkeys{SR}{SR@}{caption,first,last}{}
82 \define@cmdkeys{SR}{SR@}{fontsize,scale,label}{}

```

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

The following keys are used only by the StatRep package:

- **display** is a Boolean key that specifies that a Sascode or Datastep block should only be displayed.
- **program** is a Boolean key that specifies that a Sascode or Datastep block should only be written to the program file.
- **continued** is a Boolean key that specifies whether the output is a continuation of a preceding output block.
- **caption=** specifies the caption to use for a Listing or Graphic output.
- **first=** specifies the number of top lines to display in a Datastep environment.
- **last=** specifies the number of bottom lines to display in a Datastep environment.
- **fontsize=** specifies the \LaTeX font size to use in displaying output or code environment (for example, `small` or `footnotesize`).
- **scale=** specifies a factor by which to scale a Graphic image. For example, specify `scale=0.5` to scale the image to half its original size, or specify `scale=2` to scale it to double its original size.
- **label=** is used internally by the StatRep package to generate a label for \Graphic and \Listing elements.

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

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

- `linesize=` specifies the line size that is used to generate and typeset Listing output. Typical values are 80, 96, or 120.
- `width=` specifies the width to generate or display Graphic output.
- `dest=` specifies the SAS ODS destination to use for tabular output. The possible values are `listing` and `latex`. The `listing` option includes verbatim text-based tables from the ODS LISTING destination. The `latex` option includes LaTeX tabular environments that use the `sas` package (generated during the SAS run).

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

```

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

```
84 \define@cmdkeys{SR}[SR@]{type,style,dpi}{}
85 \define@cmdkeys{SR}[SR@]{firstobj,lastobj}{}
86 \define@cmdkeys{SR}[SR@]{objects,pattern,options}{}
87 \define@cmdkeys{SR}[SR@]{height,pagesize}{}

```

- `style=` specifies the ODS style to use in generating output.
- `dpi=` specifies how many dots per inch (DPI) to use in generating a graph.
- `firstobj=` specifies the first object's name to capture in an output stream.
- `lastobj=` specifies the last object's name to capture in an output stream.
- `objects=` specifies a space-separated list of object names to capture.
- `pattern=` specifies a name-matching pattern to select objects to capture.
- `options=` specifies other options for generating output.
- `height=` specifies the height of a generated graph (for example, `height=5.4in`).
- `pagesize=` specifies page size for output.
- `type=` specifies the type of output stream to initiate (`graphic` or `listing`). If the automatic program generation capability is in effect (that is, the `store=` option is specified), the `\Listing` and `\Graphic` tags automatically generate the appropriate value for this key.

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

```
88 \presetkeys{SR}{%
89   display = false,   program = false,   continued = false,
90   caption = \@empty, first    = 0,      last      = 0,
91   fontsize = \@empty, scale   = \@empty, label    = \@empty,

```

```

92 store      = \@empty, linesize = \@empty, width      = \@empty,
93 type       = \@empty, style    = \@empty, dpi        = \@empty,
94 dest       = \@empty, firstobj = \@empty, lastobj     = \@empty,
95 objects    = \@empty, pattern  = \@empty, options     = \@empty,
96 height     = \@empty, pagesize = \@empty}{}
```

3.3 Customizable Settings

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

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

```

97 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
98 %% Edit this line to point to the location of the StatRep macros.
99 \def\SRmacropath{\SRrootdir/statrep_macros.sas}
100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

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

```

\SRlinesize
\SRpagesize
\SRdpi
\SRstyle
\SRodsgraphopts
\SRlatexstyle
101 \ifSASServer
102 \def\SRrootdir{&currentpath}
103 \else
104 \def\SRrootdir{.}
105 \fi
106 \def\SRlinesize{80}
107 \def\SRpagesize{500}
108 \def\SRdpi{300}
109 \def\SRstyle{statistical}
110 \def\SRlatexstyle{statrep}
111 \def\SRodsgraphopts{}
```

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

- `\SRlinesize` specifies the default linesize to use for listing output.
- `\SRpagesize` specifies the default pagesize to use for listing output.
- `\SRdpi` specifies the default DPI setting to use for GRSEG and ODS graphics.
- `\SRstyle` specifies the default ODS output style to use for ODS graphics.

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

```
112 \def\SRgraphicdir{\SRrootdir/png}
113 \def\SRgraphtype{png}
114 \def\SRlistingdir{\SRrootdir/lst}
115 \def\SRlatexdir{\SRrootdir/tex}
116 \def\SRdefaultdests{listing}
```

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

- `\SRgraphicdir` specifies the name of the directory that contains the graphic output files that are generated by SAS. The default is `png`.
- `\SRgraphtype` specifies the type of graphics file that SAS will generate. The choices are `png` and `pdf`. The default is `png`.
- `\SRlistingdir` specifies the name of the directory that contains the listing (tabular) output files that are generated by SAS. The default is `lst`.
- `\SRlatexdir` specifies the name of the directory that contains the LaTeX (tabular) output files that are generated by SAS. The default is `tex`.
- `\SRdefaultdests` specifies the ODS destination to use for tabular output. The possible values are `listing` and `latex`. The default is `listing`.

```
117 \edef\SRprogramname{\jobname_SR.sas}
118 \edef\SRpreamblename{\jobname_SR_preamble.sas}
119 \def\SRmacroinclude{\@percentchar include "\SRmacropath" /nosource;}
120 \def\SRsasprogramline{%
121 \@percentchar cleandir(\SRrootdir, tex, tex);^^J%
122 \@percentchar cleandir(\SRrootdir, png, png);^^J%
123 \@percentchar cleandir(\SRrootdir, lst, lst);^^J%
124 }%
```

```
\SRprogramname
\SRmacroinclude
\SRprogramline
```

- `\SRprogramname` specifies the filename for the generated SAS program. The default is `\jobname_SR.sas`, where `\jobname` is usually the stem name of the \LaTeX source file.
- `\SRmacroinclude` specifies the line used in the generated SAS program to include the SAS macros that are bundled with the StatRep package. The default is `%include \SRmacropath /nosource;`.
- `\SRprogramline` specifies the first lines to include in the generated SAS program after the `\SRmacroinclude` line. The following default value calls a SAS macro that removes the contents of the listing, `tex`, and graphic di-

rectories. The directories are created with each SAS run that includes the macros themselves (via x commands).

```
125 \def\SRparindent{3em}
126 \def\SRintertext{... more data lines ...}
127 \def\SRtempfilename{sr.tmp}
128 \def\SRcontinuedname{continued}
129 \def\SRcaptionfont{\sffamily}
130 \def\SRcaptioncontinuedfont{\sffamily\itshape}
131 \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).
- \SR@encoding specifies the encoding of the SAS output text files (UTF-8 or WLATIN1), corresponding to the SASencodingOutput SAS macro variable. By default, this value is utf-8 or is derived from the inputenc package option.

3.4 Character Encoding (SASencodingOutput)

The StatRep package automatically manages character encoding to ensure compatibility between your L^AT_EX source document and the generated SAS output.

The package operates on the following priority:

1. **Automatic Detection:** The package checks if the `inputenc` package is loaded. If present, StatRep retrieves the active `LaTeX` encoding (for example, `utf8`, `latin1`, `ansinew`) and maps it to the corresponding SAS output encoding (`UTF-8` or `WLATIN1`). This ensures seamless handling of special characters and accents.
2. **Default Fallback:** If the `inputenc` package is not detected, the package defaults to `UTF-8` for all SAS output files.
3. **Manual Override:** The encoding can be manually forced using the package option `SASencodingOutput`. Supported values include `utf8` and `wlatin1`.

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

```
132 \InputIfFileExists{statrep.cfg}
133 {\PackageInfo{statrep}{Reading custom configuration file statrep.cfg}}
134 {\PackageInfo{statrep}{No custom configuration file found (statrep.cfg).%
135                        Using defaults.}}
```

Two output streams are created:

```
136 \newwrite\SR@program@stream
137 \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:

```
138 \def\SR@preambletext{/*^^J
139  This file is auto-generated by the statrep package.^^J
140  Do not edit this file or your changes will be lost.^^J
141  Edit the LaTeX file instead.^^J
142  ^^J
143  See the statrep package documentation and the file^^J
144  statrep.cfg for information on these settings.^^J
145  */^^J
146  ^^J
147 }
```

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.

```

148 \def\SR@write@preamble{
149   \immediate\openout\SR@tempfile@stream\SRpreamblename%
150   \SR@writepgm\SR@tempfile@stream{\SR@preambletext}
151   \SR@writepgm\SR@tempfile@stream{/* Set and invoke macro variable defaults. */}
152   \SR@writepgm\SR@tempfile@stream{\@percentchar let rootdir=\SRrootdir;}
153   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultstyle=\SRstyle;}
154   \SR@writepgm\SR@tempfile@stream{\@percentchar let latexstyle=\SRlatexstyle;}
155   \SR@writepgm\SR@tempfile@stream{\@percentchar let graphptype=\SRgraphptype;}
156   \SR@writepgm\SR@tempfile@stream{\@percentchar let graphicdir=\SRgraphicdir;}
157   \SR@writepgm\SR@tempfile@stream{\@percentchar let listingdir=\SRlistingdir;}
158   \SR@writepgm\SR@tempfile@stream{\@percentchar let latexdir=\SRlatexdir;}
159   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultlinesize=\SRlinesize;}
160   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultpagesize=\SRpagesize;}
161   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultdpi=\SRdpi;}
162   \SR@writepgm\SR@tempfile@stream{\@percentchar let odsgraphopts=\SRodsgraphopts;}
163   \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultdests=\SRdefaultdests;}
164   \SR@writepgm\SR@tempfile@stream{\@percentchar let SASencodingTex=\SR@encoding;}
165   \SR@writepgm\SR@tempfile@stream{
166     \SR@writepgm\SR@tempfile@stream{options nodate nonumber}
167     \SR@writepgm\SR@tempfile@stream{ls=&defaultlinesize ps=&defaultpagesize}
168     \SR@writepgm\SR@tempfile@stream{formchar='|---|+|---+=|-/ \@backslashchar<*>';}
169     \SR@writepgm\SR@tempfile@stream{
170       \SR@writepgm\SR@tempfile@stream{ods graphics on;}
171       \SR@writepgm\SR@tempfile@stream{/* Include SAS macro definitions. */}
172       \SR@writepgm\SR@tempfile@stream{\SRmacroinclude}
173       \SR@writepgm\SR@tempfile@stream{
174         \immediate\closeout\SR@tempfile@stream

```

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

The preamble is written to a separate file and not directly into the capture program so that users who copy and paste code from the L^AT_EX source to a SAS session can include only the preamble in their SAS session. Code has been added to automatically define the current file path for the Sever SASmode option.

```

175   \immediate\openout\SR@program@stream\SRprogramname
176   \SR@writepgm\SR@program@stream{\SR@preambletext}
177   \SR@writepgm\SR@program@stream{FILENAME current "&_SASPROGRAMFILE"; *** get the full
178   \SR@writepgm\SR@program@stream{\@percentchar LET fullpath = \@percentchar sysfunc(p
179   \SR@writepgm\SR@program@stream{\@percentchar LET SASToTexname = \@percentchar scan(
180   \SR@writepgm\SR@program@stream{\@percentchar IF \@percentchar index(&SASToTexname,
181   \SR@writepgm\SR@program@stream{\@percentchar LET SASToTexname=\@percentchar substr(
182   \SR@writepgm\SR@program@stream{\@percentchar END;}
183   \SR@writepgm\SR@program@stream{\@percentchar let currentpath = \@percentchar substr
184   \SR@writepgm\SR@program@stream{\@percentchar length(\@percentchar scan(&fullpath, -
185   \SR@writepgm\SR@program@stream{\@percentchar put &fullpath &currentpath;}
186   \SR@writepgm\SR@program@stream{\@percentchar let rc = \@percentchar sysfunc(dlgcdir

```

```

187 \SR@writepgm\SR@program@stream{\@percentchar LET SASToTexname = &SASToTexname._VarS
188 \SR@writepgm\SR@program@stream{\@percentchar put &SASToTexname;}
189 \SR@writepgm\SR@program@stream{ FILENAME outtex "&SASToTexname" ENCODING="utf-8";}
190 \SR@writepgm\SR@program@stream{ DATA _NULL_; FILE outtex LRECL=32767; RUN;}
191 \SR@writepgm\SR@program@stream{ FILENAME outtex CLEAR;}
192 \SR@writepgm\SR@program@stream{\@percentchar include "\SRrootdir\SRpreamblename" /
193 \SR@writepgm\SR@program@stream{ /* Remove all output files. */}
194 \SR@writepgm\SR@program@stream{\SRsasprogramline}
195 \SR@writepgm\SR@program@stream{
196 \SR@writepgm\SR@program@stream{ /* Start program with a null title. */}
197 \SR@writepgm\SR@program@stream{title;}
198 \SR@writepgm\SR@program@stream{
199 }

```

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

```

200 \AtBeginDocument{\SR@write@preamble}
201 \AtEndDocument{\immediate\closeout\SR@program@stream}

```

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

3.5 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.

3.5.1 Daststep Environment

Three macros are defined to set up the Daststep environment:

```
202 \def\SR@datastep@writefile#1{%
203   \setcounter{SR@totallines}{0}%
204   \@bsphack\immediate\openout\SR@tempfile@stream#1%
205   \let\do\@makeother\dospecials\catcode`\~M\active%
206   %
207   \def\verbatim@processline{%
208     \addtocounter{SR@totallines}{1}%
209     \SR@writepgm\SR@tempfile@stream{\the\verbatim@line}%
210   }%
211   \verbatim@start%
212 }
```

`\SR@datastep@writefile` The `\SR@datastep@writefile` macro is used at the beginning of the Daststep 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).

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

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

```
214 \newcommand{\SR@datastep@writeprogramline}{%
215   \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 Daststep 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 Daststep environment.

Figure 2 shows how a Daststep 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.

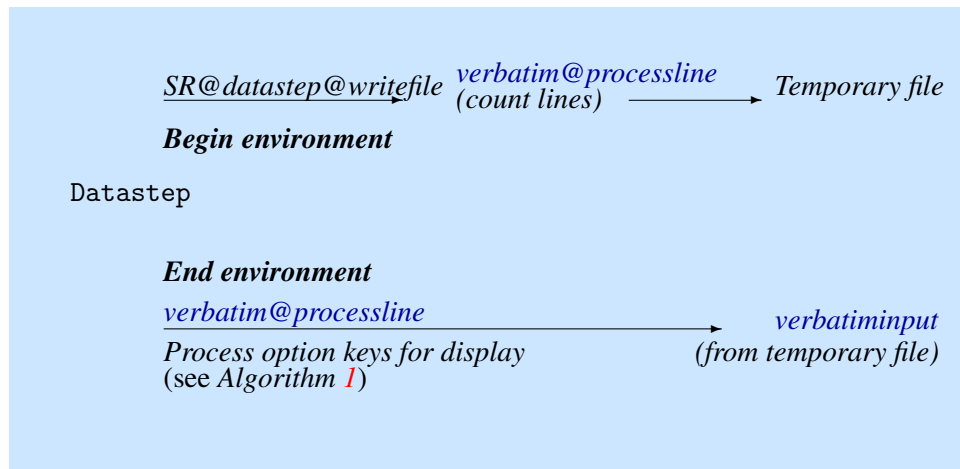


Figure 2: Processing Schematic for the Datastep Environment

```

216 \def\@Datastep[#1]{%
217   \setkeys{SR}{#1}%
218   \SR@datastep@writefile{\SRtempfilename}%
219 }
220 \newenvironment{Datastep}{%
221   \catcode\^^M=\active
222   \@ifnextchar[{\catcode\^^M=5\@Datastep}{\catcode\^^M=5\@Datastep[]}]
223 }

```

Datastep (env.) 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.

```

224 {%
225   \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.

```
226 \ifSR@program\def\verbatim@processline{\SR@datastep@writeprogramline}%
227 \else%
```

Otherwise, the code block is to be displayed.

```
228 \ifthenelse{\equal{\SR@fontsize}{\@empty}}{%
229 {\relax}%
230 {\@nameuse{\SR@fontsize}}%
231 \setcounter{SR@currentline}{0}%
232 \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.

```
233 \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.

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

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

Supporting the `last=` option in the `Datastep` environment is the only reason for the complexity of writing to a temporary file and retrieving the lines from the file. To know how to display the last n lines, `StatRep` must know the total number of lines in the code block. So it writes the code block to the temporary file and keeps a tally of the number of lines as it writes each line.

A horizontal skip of length `\SR@parindent` 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:

```

235     \ifnum\SR@first>0%
236     \ifnum\theSR@currentline<\SR@first%
237     {\leavevmode\hskip\SRparindent\the\verbatim@line\par}%
238     \else%
239     \ifnum\theSR@currentline=\SR@first%
240     {\leavevmode\par\hskip\SRparindent\SRintertext\par}%
241     \else%
242     \ifnum\SR@last>0%
243     \ifnum\theSR@currentline=\theSR@startinglastline{\leavevmode\par}\fi%
244     \ifnum\theSR@currentline>\theSR@startinglastline%
245     {\leavevmode\hskip\SRparindent\the\verbatim@line\par}%
246     \fi%
247     \fi%
248     \fi%
249     \fi%
250     \else%
251     {\leavevmode\hskip\SRparindent\the\verbatim@line\par}%
252     \fi%
253     \addtocounter{SR@currentline}{1}%
254 }%
255 \fi%

```

The following algorithm describes how this redefined version of the `\verbatim@processline` macro processes each line.

```

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

```

Algorithm 1: Datasstep 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 Datasstep 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.

```
256 \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:

```
257 \ifSR@program
258   \setlength{\SR@scratchlength}{-2\topsep-\partopsep-2\parskip-\baselineskip}
259   \vspace*{\SR@scratchlength}%
260 \fi%
```

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

```
261 \ifSR@display\else%
262   \SR@writepgm\SR@program@stream{}%
263 \fi%
264 }
```

3.5.2 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).

```
265 \begingroup
266   \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.

```

267 \gdef\S@SR@sascode@addtoline{\catcode`\%12
268     \expandafter\relax
269     \expandafter\S@SR@sascode@filter
270     \expandafter{\number\catcode`\% }}

```

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

`\relax\S@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 pdf \LaTeX : 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 pdf \TeX 1.30.0 (Thanh et al. 2009). The result of `\pdfmatch` is stored in an array.

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

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

`%* program 4;`

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

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

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

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

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

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

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

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

```

271 \gdef\SR@sascode@filter#1#2{
272   \ifnum\pdfmatch{~%\string\*}
273     {\detokenize{#2}}=1
274     \ifnum\pdfmatch{~%\string\*\space*program\space*([0-9]+);}
275       {\detokenize{#2}}=1
276       \setcounter{SR@programline}
277       {\expandafter\strip@prefix\pdflastmatch 1}
278       \catcode`~%#1\@tempwafalse\verbatim@line{ }
279     \else
280       \ifnum\pdfmatch{~%\string\*\space*display\space*([0-9]+);}
281         {\detokenize{#2}}=1
282         \setcounter{SR@displayline}
283         {\expandafter\strip@prefix\pdflastmatch 1}
284         \catcode`~%#1\@tempwafalse\verbatim@line{ }
285       \else
286         \ifnum\pdfmatch{~%\string\*\space*;(.*)$}
287           {\detokenize{#2}}=1
288           \SR@writepgm\SR@program@stream{\expandafter\strip@prefix\pdflastmatch 1}
289           \catcode`~%#1\@tempwafalse\verbatim@line{ }
290         \else \catcode`~%#1\SR@sascode@writeline{#2}

```

```

291         \fi
292     \fi
293     \fi
294 \else
295     \catcode`\%#1\SR@sascode@writeline{#2}
296     \fi
297 }
298 \endgroup

```

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

In the `\SR@sascode@filter` macro, the Boolean switch `\tempswa` must be set to false for skipped lines because of how the `verbatim` package uses the switch. In the `\verbatim@processline` macro, the switch determines whether to insert a line ending. That is, it decides whether a new `verbatim` environment is starting (`\@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.

```

299 \newcommand{\SR@sascode@writeline}[1]{%
300   \ifSR@program%
301     \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
302     \@tempswafalse%
303   \else%
304     \ifSR@display%
305       \verbatim@line\expandafter{\the\verbatim@line\hskip\SRparindent#1}%
306     \else%

```

```

307     \ifnum\theSR@programline>0%
308         \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
309         \addtocounter{SR@programline}{-1}%
310         \@tempwafalse%
311     \else%
312         \ifnum\theSR@displayline>0%
313             \verbatim@line\expandafter{\the\verbatim@line\hskip\SR@parindent#1}%
314             \addtocounter{SR@displayline}{-1}%
315         \else%
316             \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
317             \verbatim@line\expandafter{\the\verbatim@line\hskip\SR@parindent#1}%
318         \fi%
319     \fi%
320 \fi%
321 \fi%
322 }

```

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

1	for <i>each line received</i> do
2	if <i>program only</i> then
3	write line to program file
4	else if <i>display only</i> then
5	display line
6	else if <i>Program counter set</i> then
7	write line to program file;
8	decrement \SR@program counter;
9	else if <i>Display counter set</i> 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.

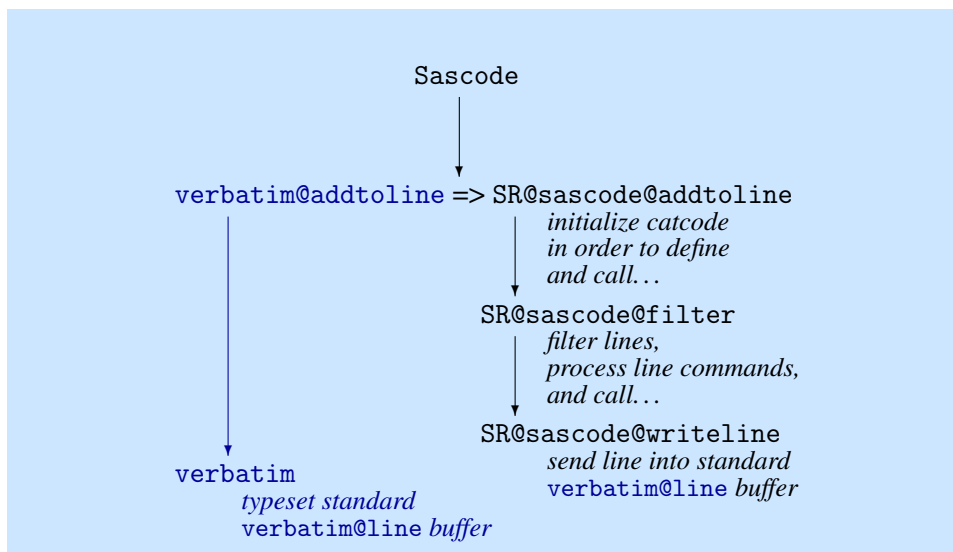


Figure 3: Processing Schematic for the Sascode Environment

Sascode (env.) 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.

```

323 \def\@Sascode[#1]{%
324   \setkeys{SR}{#1}%
325   \ifSR@display\else%
326     \ifthenelse{\equal{\SR@store}{\@empty}}{%
327       {\relax}%
328       {\SR@writepgm\SR@program@stream{\@percentchar output(\SR@store)}}%
329       \PackageInfo{statrep}{Processing Sascode block \SR@store}}%
330   \fi%
331   \ifthenelse{\equal{\SR@fontsize}{\@empty}}{%
332     {\relax}%
333     {\@nameuse{\SR@fontsize}}%
334   \setcounter{SR@programline}{0}%
335   \setcounter{SR@displayline}{0}%
336   \let\verbatim@addtoline\SR@sascode@addtoline%
337   \verbatim}
338 \newenvironment{Sascode}{%
339   \catcode`\^^M=\active
340   \@ifnextchar [{\catcode`\^^M=5\@Sascode}{\catcode`\^^M=5\@Sascode[]}]
341 }

```

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:

```

342 {\endverbatim%
343   \setlength{\SR@scratchlength}{-2\topsep-2\partopsep-2\parskip}
344   \vspace*{\SR@scratchlength}%
345   \ifSR@program\vspace*{-\baselineskip}\fi
346   %
347   \ifSR@display\else%
348     \ifthenelse{\equal{\SR@store}{\@empty}}{%
349       {\relax}%
350       {\SR@writepgm\SR@program@stream{\@percentchar endoutput(\SR@store)}}%
351       \SR@writepgm\SR@program@stream{}}%
352   \fi%
353 }

```

3.6 Handling the SAS Output

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

There are three possible cases for an output request:

- The output is of type `graphic`.
- The output is of type `listing` and destination `listing`.
- The output is of type `listing` and destination `latex`.

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

Graphical output is inserted as specified in the `\Graphic` tag, captioned, and centered.

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

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

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

`\SR@write@outoptions` is called when the `store=` option is specified in the output tag. The command parses the options that are specified in the output tag for program-related options and writes the appropriate SAS macro to the generated program.

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

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

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

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

```
354 \def\SR@write@outoptions{
355     \SR@set@outoptions
356     \let\#\SR@hashchar
```

```

357 \SR@writepgm\SR@program@stream{%
358 \percentchar write(\SR@label,store=\SR@store \SR@final@outoptions)
359 }

```

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

```

360 \SR@writepgm\SR@program@stream{}
361 }

```

\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.

```

362 \newcommand*{\SR@set@outoptions}{%
363 \global\edef\SR@final@outoptions{
364 \SR@outkeyval{height}\SR@outkeyval{objects}\SR@outkeyval{pattern}
365 \SR@outkeyval{style}\SR@outkeyval{dpi}\SR@outkeyval{options}
366 \SR@outkeyval{pagesize}\SR@outkeyval{linesize}\SR@outkeyval{width}
367 \SR@outkeyval{dest}\SR@outkeyval{type}\SR@outkeyval{firstobj}\SR@outkeyval{lastobj}
368 }

```

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.

```

369 \newcommand*{\SR@outkeyval}[1]{%
370 \def\SR@outkey{#1}
371 \edef\SR@outval{\csname SR@#1\endcsname}
372 \ifx\@empty\SR@outval
373 \else
374 \edef\SR@final@outoptions{\SR@final@outoptions,\SR@outkey=\SR@outval}
375 \fi
376 }

```

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.

3.6.1 Note on Widths

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

If the output is a graphic, the output is put into a box at whatever scale or width the user specified. The width of that box is calculated and the output centered by adjusting the `\Lleft` margin. The width calculation cannot be done at the `\Graphic` command since the output must be measured first. It is in the `\Boxgraphic` macro that the calculation takes place, which means it happens for each graphic in the output stream. Only the first graphic has an effect; it is only the first graphic in the stream that affects the `\Lleft` margin. This is an inefficiency.

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

3.6.2 The `\Listing` Macro

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

```
377 \newcommand{\Listing}[2] [] {%
378   \setkeys{SR}{#1,type=listing,label=#2}%
379   \ifthenelse{\equal{\SR@store}{\@empty}}{%
380     {\relax}%
381     {\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.

```
382 \ifthenelse{\equal{\SR@dest}{\@empty}}{%
383   {\def\SRC@dest{\SRC@defaultdests}}%
384   {\relax}%
```

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

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

```
385 \ifthenelse{\equal{\SR@dest}{listing}}%
386 %   The following statements set the font size:
387 %   \begin{macrocode}
388 {\ifthenelse{\equal{\SR@fontsize}{\@empty}}%
389   {\def\SR@fontsize{normalsize}}%
390   {\relax}}%
```

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

```
391   \settowidth{\SR@scratchlength}{\@nameuse\SR@fontsize%
392                                     \SRverbfont\selectfont~}%
```

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

```
393 \ifthenelse{\equal{\SR@linesize}{\@empty}}%
394   {\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.

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

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

```
396   \setlength{\SR@scratchlength}{\SR@verbwidth+2\tabcolsep+
397                                     2\arrayrulewidth}%
```

¹A 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).

```

398   }% end if dest is listing
399   {\relax}% do nothing if dest is latex.

```

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

```

400   \SR@insert{#2}%
401   \setlength{\LFleft}{0in}%
402 }

```

3.6.3 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).

```

403 \newcommand{\Graphic}[2] [] {%
404   \setkeys{SR}{#1,type=graphic, label=#2}%
405   \ifthenelse{\equal{\SR@store}{\@empty}}{%
406     {\relax}%
407     {\SR@write@outoptions}%
408   \SR@insert{#2}%
409   \setlength{\LFleft}{0in}%
410 }

```

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

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

3.6.4 The `\Boxlisting` Macro

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

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

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

```

411 \newcommand*{\Boxlisting}[1] {%

```

```

412 \ifthenelse{\equal{\SR@dest}{listing}}{%
413     {\global\def\S@insertname{lst/#1.lst}}%
414     \IfFileExists{\SR@insertname}%
415     {%
416         \savebox{\SR@filebox}{%
417             \def\verbatim@font{%
418                 \@nameuse{\SR@fontsize}%
419                 \SRverbfont%
420                 \hyphenchar\font\m@ne%
421                 \@noligs%
422             }%
423             \parbox{\SR@verbwidth}{%
424                 \vspace*{0.25\baselineskip}%
425                 \noindent\verbatiminput{\SR@insertname}%
426                 \vspace*{0.25\baselineskip}}%
427             }%
428         }%
429         {\savebox{\SR@filebox}{\fbox{Missing File \SR@insertname}}}%
430         \global\def\S@insertname{@empty}}}%
431 }%
432 {\global\def\S@insertname{tex/#1.tex}}%
433 \IfFileExists{\SR@insertname}%
434 {%
435     \savebox{\SR@filebox}{%
436         \begin{minipage}{\textwidth}%
437             \vspace{0.25\baselineskip}%
438             \centering\input{\SR@insertname}%
439             \vspace{0.25\baselineskip}%
440         \end{minipage}%
441     }%
442 }%
443 {\savebox{\SR@filebox}{\centering\fbox{Missing File \SR@insertname}}}%
444 \global\def\S@insertname{@empty}%
445 }%
446 }%
447 \global\setbox\S@filebox=\box\S@filebox%
448 \PackageInfo{statrep}{Processing output \SR@insertname}%
449 }

```

`\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 3.6.2. White space at the top and bottom of the box is inserted (`0.25\baselineskip`).

The `\verbatim@font` is set to the user-defined `\SR@fontsize` (from the options on the Listing tag) and the `\SRverbfont` in the configuration. Furthermore, ligatures and the hyphenation character are both turned off. If you omit this step,

ligatures will appear in the listing output (the case which prompted this bug-fix was a dashed line; the emdash ligature was applied to each pair of dashes).

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

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

3.6.5 The `\Boxgraphic` Macro

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

```

450 \newcommand*{\Boxgraphic}[1]{%
451   \IfFileExists{png/#1.\SRgraphtype}%
452     {\global\def\SRIinsertname{png/#1.\SRgraphtype}}%
453     {\global\def\SRIinsertname{\@empty}}%
454   \ifthenelse{\equal{\SRIinsertname}{\@empty}}%
455     {\savebox{\SR@filebox}{\fbox{ Missing File png/#1.\SRgraphtype}}}%
456     {%
457       \ifthenelse{\equal{\SR@scale}{\@empty}}%
458         {\ifthenelse{\equal{\SR@width}{\@empty}}%
459           {\savebox{\SR@filebox}{\includegraphics{\SRIinsertname}}%
460           {\savebox{\SR@filebox}{\includegraphics[width=\SR@width]%
461                                     {\SRIinsertname}}}}%
462           {\savebox{\SR@filebox}{\includegraphics[scale=\SR@scale]%
463                                     {\SRIinsertname}}}}%
464       }%
465   \global\setbox\SRIfilebox=\box\SRIfilebox%
466   \PackageInfo{statrep}{Processing graphic \SRIinsertname}%
467   \setlength{\SR@scratchlength}{\wd\SRIfilebox+2\tabcolsep}%
468 }%
```

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

The macro first checks whether the file exists. If it does, it sets the global variable `\SR@insertname` to the full name. If the file does not exist, `\SR@insertname` is defined to be `\@empty`.

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

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

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

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

3.6.6 The `\SR@insert` Macro

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

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

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

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

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

```
469 \def\S@set@outheadings{%
```

The following statements define three macros for typing shortcuts:

```
470 \def\S@caption{\caption{\SRcaptionfont\S@caption}}%
471 \def\S@captioncontinued{\caption{\SRcaptioncontinuedfont\S@continuedname}}%
472 \def\S@captionskip{\setlength{\baselineskip}{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:

```
473 \ifthenelse{\equal{\SR@dest}{listing}}{%
474     {\def\S@@firsthead{\hline\endLFfirsthead}%
475     \def\S@@conthead{\hline\endLFhead}%
476     \def\S@@endfoot{\hline\endLFfoot}}%
477     %
478     {\def\S@@firsthead{\endLFfirsthead}%
479     \def\S@@conthead{\endLFhead}%
480     \def\S@@endfoot{\endLFfoot}}%
481     %%%
```

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

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

```

482   %%%
483   \renewcommand*{\SR@firsthead}{\SR@@firsthead}%
484   \renewcommand*{\SR@conthead}{\SR@@conthead}%
485   \renewcommand*{\SR@endfoot}{\SR@@endfoot}%

486   \ifthenelse{\equal{\SR@caption}{\@empty}}%
487     {% CAPTION NOT GIVEN
488       \addtocounter{\LFcounter}{-1}%
489       \ifSR@continued%
490         \renewcommand*{\SR@conthead}{\SR@@captioncontinued%
491                               \SR@@captionskip\SR@@conthead}%
492       \fi%
493     }%

```

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.

```

494     {% CAPTION GIVEN
495       \renewcommand*{\SR@firsthead}{\SR@@caption\label{\SR@label}%
496                                   \SR@@captionskip\SR@@firsthead}%
497       \renewcommand*{\SR@conthead}{\SR@@captioncontinued%
498                                   \SR@@captionskip\SR@@conthead}%
499     }
500 }

```

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

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

```

501 \def\S@set@outmargin{%
502   \ifdim\S@scratchlength>\textwidth%
503     \dosloppy%
504     \setlength{LFleft}{(\paperwidth-\SR@scratchlength)/\real{2.0}}%
505     \addtolength{LFleft}{-1in}%
506     \addtolength{LFleft}{-\hoffset}%
507     \addtolength{LFleft}{-\leftskip}%
508     \ifodd\c@page\addtolength{LFleft}{-\oddsidemargin}%
509     \else\addtolength{LFleft}{-\evensidemargin}%

```

```

510     \fi%
511   \else%
512     \setlength{\Lleft}{(\textwidth-\SR@scratchlength)/\real{2.0}}%
513   \fi%
514 }

```

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 T_EX margin (1 inch)
- `\hoffset`
- `\leftskip`
- the margin specified in the document design

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

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

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

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 `\hoffset` and `\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 `\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:

```
515 \newcommand{\SR@insert}[1]{%
516   \SR@set@outheadings%
517   \ifthenelse{\equal{\SR@type}{listing}}{\Boxlisting{#1}}{\Boxgraphic{#1}}%
518   \ifthenelse{\equal{\SR@dest}{latex}}%
519     {\relax}%
520     {\SR@set@outmargin}%
```

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

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

```
521   \ifthenelse{\equal{\SR@insertname}{\@empty}}%
522     {\setboolean{SR@multifile}{false}}%
523     {\setboolean{SR@multifile}{true}}%
```

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

```
524   \ifthenelse{\equal{\SR@dest}{listing}}%
525     {\bgroup\longfigure{|c|}}{\bgroup\longfigure{c}}%
526     \SR@firsthead\SR@conthead\SR@endfoot%
527     \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:

```
528     \setcounter{SR@multifilecount}{1}%
529     \whiledo{\boolean{SR@multifile}}{%
530       \ifthenelse{\equal{\SR@type}{listing}}%
531         {\Boxlisting{#1\arabic{SR@multifilecount}}}%
532         {\Boxgraphic{#1\arabic{SR@multifilecount}}}%
533       \ifthenelse{\equal{\SR@insertname}{\@empty}}%
534         {\setboolean{SR@multifile}{false}}%
535         {\[0.5\baselineskip]\usebox{\SR@filebox}%
536         \stepcounter{SR@multifilecount}%
537         \setboolean{SR@multifile}{true}}%
538     }%
539   \endlongfigure\egroup%
```

```

540 \unsloppy%
541 }

```

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

1	while <i>multiple files is True</i> do
2	if <i>type=listing</i> then
3	Boxlisting
4	else
5	Boxgraphic
6	end
7	if <i>file exists</i> 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.

3.7 SAS Result Export Macros

The StatRep package provides SAS macros designed to export numeric and date variables directly into L^AT_EX documents in a consistent and properly formatted way.

- Predefined proc format picture formats handle small numbers, large numbers, percentages, and dates.
- %STATREResults(variable, format) writes a single variable to a L^AT_EX file as a \def command using the chosen format.

- `%STATREPA11Results()` automatically processes all numeric variables in a dataset and generates their \LaTeX definitions with appropriate formatting.
- `%STATREPMACROVAR(variable)` provides a convenient way to export one variable using `%STATREPA11Results`.
- All macros write the output to a file named `\jobname_SASToTex.tex`, which can be included in your main \LaTeX document with `\input`.

These macros make it easy to synchronize SAS results with \LaTeX reports, ensuring consistency in number formatting, dates, and scientific notation across your documents.

```
% title 'Probit Analysis, Newspaper Survey';
% proc format;
%   value subscrib 1 = 'accept' 0 = 'reject';
% run;
% data news;
%   input sex $ 1-6 age 12-13 subs 18 ;
%   datalines;
% Female      35    0
% Male        45    1
%   ... more data lines ...
% Female      32    0
% Female      35    0
% ;
%
% %LET PARAMETER=365;
% %STATREPALLRESULTS(PARAMETER)
%
```

Figure 4: Example of `%STATREPALLRESULTS` Usage

The file `_VarSASToTex.tex` includes the command `\def\PARAMETER{365}`, which can be used in the \LaTeX document as `\PARAMETER` (see also section 3.7).
<*longfigure>

4 longfigure Usage

The `longfigure` package uses and relabels components of the well-known `longtable` package, written by David Carlisle, to provide a table-like environment that can display a stream of subfigures as a single figure that can break across pages.

The `longtable` package defines a `longtable` environment, which produces tables that can be broken by \TeX 's standard page-breaking algorithm. Similarly,

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

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

- The `longfigure` package supports two additional key-value options:
 - The `figname=` option specifies the counter for numbering `longfigure` environments. You can specify any string; the default is `figure`. When you specify a `figname=` value for which no counter already exists, the `longfigure` package loads the `tocloft` package and creates the counter.
 - The `resetby=` option specifies a counter (for example, `resetby=chapter`) such that output numbering is reset each time the counter value changes. If a counter is specified that does not exist, the `tocloft` package is loaded to create the new counter. For information about how the lists are typeset, see the `tocloft` package documentation.
- The counters and macros that start with `\LT` in the `longtable` package are renamed to start with `\LF` in the `longfigure` package to avoid namespace conflicts when the two packages are used together. The generic macros that are defined in the `longtable` package (`\endfirsthead`, `\endhead`, `\endfoot`, and `\endlastfoot`) are also renamed with `\LF` as a prefix in the `longfigure` package.
- The `\LF@name` macro is based on the `\fnum@table` macro from the `longtable` package. The `\LF@name` macro returns the capitalized counter name and value. For example, if the counter is `figure` and the macro is processing the second `longfigure`, the `\LF@name` macro would contain the value “Figure 2.”

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

```
\listoffigures
```

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

```
\listofdisplay
```


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

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

4.1 Example

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

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

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

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

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

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

\begin{document}
  \begin{longfigure}{c}
    \caption{My Long Figure}\label{mlfig2}\\
  \end{longfigure}
\end{document}
```

```

\hline\endLFfirsthead
\caption{continued}\\
\hline\endLFhead
\hline\endLFfoot
\hline\hline\endLFastfoot
\includegraphics[width=3in]{myfig1}\\
\includegraphics[width=3in]{myfig2}\\
\includegraphics[width=3in]{myfig3}\\
\begin{tabular}{ll}
    one & two \\
    three & four\\
\end{tabular}
\end{longfigure}
\end{document}

```

5 Implementation

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

```
542 \ProvidesPackage{longfigure}[2014/01/06 longfigure]
```

The following statement loads the `xkeyval` package for declaring and processing package options:

```
543 \RequirePackage{xkeyval}
```

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

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

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

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

5.1 Options

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

```

546 \DeclareOptionX{figname}[figure]{\renewcommand*{\LFcounter}{#1}}
547 \DeclareOptionX{resetby}{\renewcommand*{\LFreset}{#1}}

```

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

```

548 \DeclareOptionX{set}{}
549 \DeclareOptionX{final}{}
550 \DeclareOptionX{errorshow}{\def\LF@warn{\PackageInfo{longfigure}}}
551 \DeclareOptionX{pausing}{\def\LF@warn#1{\LF@err{#1}{This is not really an error}}}
552 \ProcessOptionsX

```

The following statements process the options:

```

553 \def\LFProcessOptions#1{
554   \@ifundefined{c@#1}{%
555     \RequirePackage{tocloft}
556     \def\LFuc##1##2{\MakeUppercase{##1}{##2}}
557     \expandafter\def\csname list#1name\endcsname{List of \LFuc#1s}
558     \ifx\@empty\LFreset%
559       \newlistof{#1}{lft}{\csname list#1name\endcsname}
560     \else
561       \newlistof[\LFreset]{#1}{lft}{\csname list#1name\endcsname}
562     \fi
563   }{}%
564 }
565 \expandafter\LFProcessOptions\expandafter{\LFcounter}

```

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

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

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

5.2 Utilities

`\strcfstr` The following macro, `\strcfstr`, checks whether two strings, which are provided as arguments, are equal (Wilson, 2001). A new boolean `\ifLF@same` contains the result of the test.

```

566 \newif\ifLF@same
567 \newcommand{\strcfstr}[2]{%
568   \LF@samefalse
569   \begingroup\def\2{#2}

```

```

570 \ifx\2#1\endgroup\LF@sametrue
571 \else\endgroup
572 \fi
573 }

```

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

```

574 \def\LFupcase#1{%
575 \def\x##1##2{%
576 \MakeUppercase{##1}{##2}}\x#1%
577 }

```

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

```

578 \def\LF@name{\expandafter\LFupcase%
579 \expandafter{\LFcounter}~%
580 \expandafter\csname the\LFcounter\endcsname}%

```

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

```

581 \def\LF@err{\PackageError{longfigure}}
582 \def\LF@warn{\PackageWarning{longfigure}}
583 \def\LF@final@warn{%
584 \AtEndDocument{%
585 \LF@warn{\LFcounter \@width s have changed. Rerun \LaTeX\.\@gobbletwo}}%
586 \global\let\LF@final@warn\relax
587 %
588 \newskip\LFleft \LFleft=\fill
589 \newskip\LFright \LFright=\fill
590 \newskip\LFpre \LFpre=\bigskipamount
591 \newskip\LFpost \LFpost=\bigskipamount
592 \newcount\LFchunksize \LFchunksize=20
593 \let\c@LFchunksize\LFchunksize
594 \newdimen\LFcapwidth \LFcapwidth=4in
595 \newbox\LF@head
596 \newbox\LF@firsthead
597 \newbox\LF@foot
598 \newbox\LF@lastfoot
599 \newcount\LF@cols
600 \newcount\LF@rows
601 \newcounter{LF@tables}
602 \newcounter{LF@chunks}[LF@tables]
603 %

```

```

604 \newtoks\LF@p@ftn
605 \mathchardef\LF@end@pen=30000
606 \def\longfigure{%
607   \par
608   \ifx\multicols\@undefined
609   \else
610     \ifnum\col@number>\@ne
611     \@twocolumntrue
612     \fi
613   \fi
614   \if@twocolumn
615     \LF@err{longfigure not in 1-column mode}\@ehc
616   \fi
617   \begingroup
618   \@ifnextchar[\LF@array{\LF@array[x]}
619 \def\LF@array[#1]#2{%
620   \refstepcounter{\LF@counter}\stepcounter{\LF@tables}%
621   \if l#1%
622     \LFleft\z@ \LFright\fill
623   \else\if r#1%
624     \LFleft\fill \LFright\z@
625   \else\if c#1%
626     \LFleft\fill \LFright\fill
627   \fi\fi\fi
628   \let\LF@mcol\multicolumn
629   \let\LF@@tabarray\@tabarray
630   \let\LF@@hl\hline
631   \def\@tabarray{%
632     \let\hline\LF@@hl
633     \LF@@tabarray}%
634   \let\\ \LF@tabularcr\let\tabularnewline\\
635   \def\newpage{\noalign{\break}}%
636   \def\pagebreak{\noalign{\ifnum`}=0\fi\@testopt{\LF@no@pgbk-}4}%
637   \def\nopagebreak{\noalign{\ifnum`}=0\fi\@testopt{\LF@no@pgbk4}}%
638   \let\hline\LF@hline \let\kill\LF@kill\let\caption\LF@caption
639   \@tempdima\ht\strutbox
640   \let\@endpbox\LF@endpbox
641   \ifx\extrarowheight\@undefined
642     \let\@acol\@tabacol
643     \let\@classz\@tabclassz \let\@classiv\@tabclassiv
644     \def\@startpbox{\vtop\LF@startpbox}%
645     \let\@@startpbox\@startpbox
646     \let\@@endpbox\@endpbox
647     \let\LF@LL@FM@cr\@tabularcr
648   \else
649     \advance\@tempdima\extrarowheight
650     \col@sep\tabcolsep
651     \let\@startpbox\LF@startpbox\let\LF@LL@FM@cr\@arraycr
652   \fi

```

```

653 \setbox\@arstrutbox\hbox{\vrule
654   \@height \arraystretch \@tempdima
655   \@depth \arraystretch \dp \strutbox
656   \@width \z@}%
657 \let\@sharp#\let\protect\relax
658 \begingroup
659   \@mkpream{#2}%
660   \xdef\LF@bchunk{%
661     \global\advance\c@LF@chunks\@ne
662     \global\LF@rows\z@\setbox\z@\vbox\bgroup
663     \LF@setprevdepth
664     \tabskip\LF@left \noexpand\halign to\hsize\bgroup
665     \tabskip\z@ \@arstrut \@preamble \tabskip\LF@right \cr}%
666 \endgroup
667 \expandafter\LF@nofcols\LF@bchunk&\LF@nofcols
668 \LF@make@row
669 \m@th\let\par\@empty
670 \everycr{}\lineskip\z@\baselineskip\z@
671 \LF@bchunk}
672 \def\LF@no@pgbk#1[#2]{\penalty #1\@getpen{#2}\ifnum`{=0\fi}}
673 \def\LF@start{%
674   \let\LF@start\endgraf
675   \endgraf\penalty\z@\vskip\LF@pre
676   \dimen@ \pagetotal
677   \advance\dimen@ \ht\ifvoid\LF@firsthead\LF@head\else\LF@firsthead\fi
678   \advance\dimen@ \dp\ifvoid\LF@firsthead\LF@head\else\LF@firsthead\fi
679   \advance\dimen@ \ht\LF@foot
680   \dimen@ii\vfuzz
681   \vfuzz\maxdimen
682   \setbox\tw@\copy\z@
683   \setbox\tw@\vsplit\tw@ to \ht\@arstrutbox
684   \setbox\tw@\vbox{\unvbox\tw@}%
685   \vfuzz\dimen@ii
686   \advance\dimen@ \ht
687     \ifdim\ht\@arstrutbox>\ht\tw@\@arstrutbox\else\tw@\fi
688   \advance\dimen@\dp
689     \ifdim\dp\@arstrutbox>\dp\tw@\@arstrutbox\else\tw@\fi
690   \advance\dimen@ -\pagegoal
691   \ifdim \dimen@>\z@\vfil\break\fi
692   \global\@colroom\@colht
693   \ifvoid\LF@foot\else
694     \advance\vsizel-\ht\LF@foot
695     \global\advance\@colroom-\ht\LF@foot
696     \dimen@\pagegoal\advance\dimen@-\ht\LF@foot\pagegoal\dimen@
697     \maxdepth\z@
698   \fi
699   \ifvoid\LF@firsthead\copy\LF@head\else\box\LF@firsthead\fi\nobreak
700   \output{\LF@output}}
701 \def\endlongfigure{%

```

```

702 \crrc
703 \noalign{%
704   \let\LF@entry\LF@entry@chop
705   \xdef\LF@save@row{\LF@save@row}}%
706 \LF@echunk
707 \LF@start
708 \unvbox\z@
709 \LF@get@widths
710 \if@filesw
711   {\let\LF@entry\LF@entry@write\immediate\write\@auxout{%
712     \gdef\expandafter\noexpand
713       \csname LF@\romannumeral\c@LF@tables\endcsname
714         {\LF@save@row}}}%
715   \fi
716   \ifx\LF@save@row\LF@@@save@row
717   \else
718     \LF@warn{Column \@width s have changed\MessageBreak
719       in table \thetable}%
720     \LF@final@warn
721   \fi
722 \endgraf\penalty -\LF@end@pen
723 \endgroup
724 \global\@mparbottom\z@
725 \pagegoal\vsizer
726 \endgraf\penalty\z@\advspace\LFpost
727 \ifvoid\footins\else\insert\footins{}\fi}
728 \def\LF@nofcols#1&{%
729   \futurelet\@let@token\LF@n@fcols}
730 \def\LF@n@fcols{%
731   \advance\LF@cols\@ne
732   \ifx\@let@token\LF@nofcols
733     \expandafter\@gobble
734   \else
735     \expandafter\LF@nofcols
736   \fi}
737 \def\LF@tabularcrr{%
738   \relax\iffalse{\fi\ifnum0=`}\fi
739   \@ifstar
740     {\def\crrc{\LF@crrc\noalign{\nobreak}}\let\cr\crrc
741       \LF@t@bularcrr}%
742     {\LF@t@bularcrr}}
743 \let\LF@crrc\crrc
744 \let\LF@setprevdepth\relax
745 \def\LF@t@bularcrr{%
746   \global\advance\LF@rows\@ne
747   \ifnum\LF@rows=\LF@chunksize
748     \gdef\LF@setprevdepth{%
749       \prevdepth\z@\global
750       \global\let\LF@setprevdepth\relax}%

```

```

751 \expandafter\LF@xtabularcr
752 \else
753 \ifnum0=`{}\fi
754 \expandafter\LF@LL@FM@cr
755 \fi}
756 \def\LF@xtabularcr{%
757 \@ifnextchar[\LF@argtabularcr\LF@ntabularcr}
758 \def\LF@ntabularcr{%
759 \ifnum0=`{}\fi
760 \LF@echunk
761 \LF@start
762 \unvbox\z@
763 \LF@get@widths
764 \LF@bchunk}
765 \def\LF@argtabularcr[#1]{%
766 \ifnum0=`{}\fi
767 \ifdim #1>\z@
768 \unskip\@xarraycr{#1}%
769 \else
770 \@yarraycr{#1}%
771 \fi
772 \LF@echunk
773 \LF@start
774 \unvbox\z@
775 \LF@get@widths
776 \LF@bchunk}
777 \def\LF@echunk{%
778 \crr\LF@save@row\cr\egroup
779 \global\setbox\@ne\lastbox
780 \unskip
781 \egroup}
782 \def\LF@entry#1#2{%
783 \ifhmode\@firstofone{&}\fi\omit
784 \ifnum#1=\c@LF@chunks
785 \else
786 \kern#2\relax
787 \fi}
788 \def\LF@entry@chop#1#2{%
789 \noexpand\LF@entry
790 {\ifnum#1>\c@LF@chunks
791 1}{0pt%
792 \else
793 #1}{#2%
794 \fi}}
795 \def\LF@entry@write{%
796 \noexpand\LF@entry^^J%
797 \@spaces}
798 \def\LF@kill{%
799 \LF@echunk

```



```

800 \LF@get@widths
801 \expandafter\LF@rebox\LF@bchunk}
802 \def\LF@rebox#1\bgroup{%
803   #1\bgroup
804   \unvbox\z@
805   \unskip
806   \setbox\z@\lastbox}
807 \def\LF@blank@row{%
808   \xdef\LF@save@row{\expandafter\LF@build@blank
809     \romannumeral\number\LF@cols 001 }}
810 \def\LF@build@blank#1{%
811   \if#1m%
812     \noexpand\LF@entry{1}{0pt}%
813     \expandafter\LF@build@blank
814   \fi}
815 \def\LF@make@row{%
816   \global\expandafter\let\expandafter\LF@save@row
817     \csname LF@romannumeral\c@LF@tables\endcsname
818   \ifx\LF@save@row\relax
819     \LF@blank@row
820   \else
821     {\let\LF@entry\or
822       \if!%
823         \ifcase\expandafter\expandafter\expandafter\LF@cols
824         \expandafter@\gobble\LF@save@row
825         \or
826         \else
827         \relax
828         \fi
829         !%
830       \else
831         \aftergroup\LF@blank@row
832       \fi}%
833   \fi}
834 \let\setlongfigures\relax
835 \def\LF@get@widths{%
836   \setbox\tw@\hbox{%
837     \unhbox\@ne
838     \let\LF@old@row\LF@save@row
839     \global\let\LF@save@row\@empty
840     \count@\LF@cols
841     \loop
842       \unskip
843       \setbox\tw@\lastbox
844     \ifhbox\tw@
845       \LF@def@row
846       \advance\count@\m@ne
847     \repeat}%
848   \ifx\LF@@save@row\@undefined

```

```

849   \let\LF@@save@row\LF@save@row
850   \fi}
851 \def\LF@def@row{%
852   \let\LF@entry\or
853   \edef\@tempa{%
854     \ifcase\expandafter\count@\LF@old@row
855     \else
856       {1}{0pt}%
857     \fi}%
858   \let\LF@entry\relax
859   \xdef\LF@save@row{%
860     \LF@entry
861     \expandafter\LF@max@sel\@tempa
862     \LF@save@row}}
863 \def\LF@max@sel#1#2{%
864   {\ifdim#2=\wd\tw@
865     #1%
866   \else
867     \number\c@LF@chunks
868   \fi}%
869   {\the\wd\tw@}}
870 \def\LF@hline{%
871   \noalign{\ifnum0=}\fi
872   \penalty\@M
873   \futurelet\@let@token\LF@@hline}
874 \def\LF@@hline{%
875   \ifx\@let@token\hline
876     \global\let\@gtempa\@gobble
877     \gdef\LF@sep{\penalty-\@medpenalty\vskip\doublerulesep}%
878   \else
879     \global\let\@gtempa\@empty
880     \gdef\LF@sep{\penalty-\@lowpenalty\vskip-\arrayrulewidth}%
881   \fi
882   \ifnum0=}\fi}%
883   \multispan\LF@cols
884     \unskip\leaders\hrule\@height\arrayrulewidth\hfill\cr
885   \noalign{\LF@sep}%
886   \multispan\LF@cols
887     \unskip\leaders\hrule\@height\arrayrulewidth\hfill\cr
888   \noalign{\penalty\@M}%
889   \@gtempa}

```

5.3 Captioning

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

```

890 \def\LF@caption{%

```

```

891 \noalign\bgroup
892   \@ifnextchar[{\egroup\LF@c@ption\@firstofone}\LF@capti@n}

```

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

```

893 \def\LF@c@ption#1[#2]#3{%
894   \LF@makecaption#1\LF@name{#3}%
895   \def\@tempa{#2}%
896   \ifx\@tempa\@empty\else

```

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

The code writes to one of the following files.

- If the counter is `figure`, write to the `lof` file.
- If the counter is `table`, write to the `lot` file.
- Otherwise, write to `lft`, a file created here for this purpose.

```

897   {\let\\space
898   \strcfstr{\LFcounter}{figure}
899   \ifLF@same\def\LFoutfile{lof}\else
900     \strcfstr{\LFcounter}{table}
901     \ifLF@same\def\LFoutfile{lot}\else
902       \def\LFoutfile{lft}\fi\fi
903   \addcontentsline{\LFoutfile}{\LFcounter}
904   {\expandafter\protect\expandafter\numberline\expandafter%
905   {\expandafter\csname the\LFcounter\endcsname}{#2}}}%
906   \fi
907 }

```

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

```

908 \def\LF@capti@n{%
909   \@ifstar
910   {\egroup\LF@c@ption\@gobble[]}%
911   {\egroup\@xdblarg{\LF@c@ption\@firstofone}}}

```

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

```

912 \def\LF@makecaption#1#2#3{%

```

```

913 \LF@mc\LF@cols c{\hbox to\z@{\hss\parbox[t]\LFcapwidth{%
914 \sbox\@tempboxa{#1{#2: }#3}%
915 \ifdim\wd\@tempboxa>\hsize
916 #1{#2: }#3%
917 \else
918 \hbox to\hsize{\hfil\box\@tempboxa\hfil}%
919 \fi
920 \endgraf\vskip\baselineskip}%
921 \hss}}
922 \def\LF@output{%
923 \ifnum\outputpenalty <-\@Mi
924 \ifnum\outputpenalty > -\LF@end@pen
925 \LF@err{floats and marginpars not allowed in a longfigure}\@ehc
926 \else
927 \setbox\z@\vbox{\unvbox\@cclv}%
928 \ifdim \ht\LF@lastfoot>\ht\LF@foot
929 \dimen@ \pagegoal
930 \advance\dimen@-\ht\LF@lastfoot
931 \ifdim\dimen@<\ht\z@
932 \setbox\@cclv\vbox{\unvbox\z@\copy\LF@foot\vss}%
933 \@makecol
934 \@outputpage
935 \setbox\z@\vbox{\box\LF@head}%
936 \fi
937 \fi
938 \global\@colroom\@colht
939 \global\vsiz\@colht
940 \vbox
941 {\unvbox\z@\box\ifvoid\LF@lastfoot\LF@foot\else
942 \LF@lastfoot\fi}%
943 \fi
944 \else
945 \setbox\@cclv\vbox{\unvbox\@cclv\copy\LF@foot\vss}%
946 \@makecol
947 \@outputpage
948 \global\vsiz\@colroom
949 \copy\LF@head\nobreak
950 \fi}
951 \def\LF@end@hd@ft#1{%
952 \LF@echunk
953 \ifx\LF@start\endgraf
954 \LF@err
955 {Longfigure head or foot not at start of table}%
956 {Increase LFchunksize}%
957 \fi
958 \setbox#1\box\z@
959 \LF@get@widths
960 \LF@bchunk}

```

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

```

961 \def\endLFfirsthead{\LF@end@hd@ft\LF@firsthead}
962 \def\endLFhead{\LF@end@hd@ft\LF@head}
963 \def\endLFfoot{\LF@end@hd@ft\LF@foot}
964 \def\endLFastfoot{\LF@end@hd@ft\LF@lastfoot}
965 %
966 \def\LF@startpbox#1{%
967   \bgroup
968     \let\@footnotetext\LF@p@ftntext
969     \setlength\hsize{#1}%
970     \@arrayparboxrestore
971     \vrule \@height \ht\@arstrutbox \@width \z@}
972 \def\LF@endpbox{%
973   \@finalstrut\@arstrutbox
974   \egroup
975   \the\LF@p@ftn
976   \global\LF@p@ftn{}}%
977 \hfil}
978 \def\LF@p@ftntext#1{%
979   \edef\@tempa{\the\LF@p@ftn\noexpand\footnotetext[\the\c@footnote]}}%
980   \global\LF@p@ftn\expandafter{\@tempa{#1}}}%

```

5.4 References

- Carlisle, D. 2004. *The longtable Package*. Included in the “Comprehensive T_EX Archive Network.” <http://ctan.org>.
- Lazarides, Y. 2010. T_EXstackexchange, online forum. <http://tex.stackexchange.com/questions/7992>.
- Schöpf, R, B. Raichle, and C. Rowley. 2001. *A New Implementation of L^AT_EX’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 pdfT_EX User’s Manual,” Revision 655, corresponding to pdfT_EX 1.40.11. www.tug.org/applications/pdftex.
- Wilson, P. 2001. *Glisterings*. In TUGboat 22(4), 339–340.