The statrep package*

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

2014/6/16

Contents

*This document corresponds to statrep v1.02, last revised 2014/6/16.

List of Figures

1 About This Document

This document uses the following aids for presenting information.

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

2 StatRep Usage

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

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

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

Figure 1: Example of Using the StatRep Package

The StatRep package supports the following options:

- `generate` specifies whether a SAS program is generated at compile time. It can have a value of `true` or `false`; the default is `true`.
- `figname=` specifies the name of a LaTeX counter that is used for numbering outputs. The default is `figure`. If you specify a value for the `figname` option for which no counter exists, a counter is created.
- `resetby=` specifies that the counter for output numbering be reset with each change in the specified counter value. For example, if `resetby=chapter`, all output numbering is reset when the chapter value changes. See section 4 for details. Also, refer to the `tocloft` package documentation for information about how the lists are typeset.

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

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

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

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

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

3 StatRep Implementation

3.1 Required Packages

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

```
1 \RequirePackage{verbatim}
```

```
\RequirePackage{graphicx}
\RequirePackage{xkeyval}
\RequirePackage{calc}
\RequirePackage{ifthen}
```

- The `verbatim` package provides the foundation for the StatRep package.
- The `graphicx` package enables inclusion of images and is used by the `\Graphic` tag.
- The `xkeyval`, `calc`, and `ifthen` packages provide programming capabilities that are used throughout the StatRep package.

```
\newif\ifSR@generate\SR@generatetrue
\DeclareOptionX{generate}[true]{\@nameuse{SR@generate#1}}
\DeclareOptionX{figname}{\PassOptionsToPackage{\CurrentOption}{longfigure}}
\DeclareOptionX{resetby}{\PassOptionsToPackage{\CurrentOption}{longfigure}}
\ProcessOptionsX
\RequirePackage{longfigure}
```

3.2 Programming Utilities

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

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

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

`\SR@writepgm`

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:

```
\newcommand*{\dosloppy}{\setlength{\hfuzz}{\maxdimen}\hbadness\maxdimen}
```

```
\newcommand*{\unsloppy}{\setlength{\hfuzz}{0.3pt}\hbadness 1414}
```

\dosloppy
\unsloppy

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 LaTeX's line-breaking algorithms or line penalties; they suppress the overfull box (\hfuzz) and underfull box (\hbadness) warnings in the log. The system uses theses commands when it insert outputs. See the \SR@set@outmargin macro in section 3.5.5 for details.

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

```
\newlength{\SR@scratchlength}
\newlength{\SR@verbwidth}
\newsavebox{\SR@filebox}
\newcommand*{\SR@firsthead}{}
\newcommand*{\SR@conthead}{}
\newcommand*{\SR@endfoot}{}
```

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

```
\newcounter{SR@currentline}\setcounter{SR@currentline}{0}
\newcounter{SR@displayline}\setcounter{SR@displayline}{0}
\newcounter{SR@programline}\setcounter{SR@programline}{0}
\newcounter{SR@startinglastline}\setcounter{SR@startinglastline}{0}
\newcounter{SR@totallines}\setcounter{SR@totallines}{0}
\newcounter{SR@multifilecount}\setcounter{SR@multifilecount}{0}
\newboolean{SR@multifile}\setboolean{SR@multifile}{false}
```

- SR@currentline contains the current line number of the verbatim environment that is being processed.
- SR@displayline contains the number of lines to display. The number can be specified as a line command inside a Sascode environment.
- SR@programline contains the number of lines to write to the generated program. The number can be specified as a line command inside a Sascode environment.
- SR@startinglastline contains the line number that begins the bottom block of an abbreviated displayed Datastep environment. This counter is used when the last= option is specified in the Datastep environment.

- SR@totallines contains the total number of lines in the verbatim environment.
- SR@multifilecount counter is used to step through and insert a set of output files that are generated by SAS.
- SR@multifile is a Boolean switch that is used in the algorithm that inserts the stream of outputs. See section 3.5.5 for complete details.

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

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

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

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

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

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

The following keys are used only by the StatRep package:

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

- `continued` is a Boolean key that specifies whether the output is a continuation of a preceding output block.
- `caption=` specifies the caption to use for a `Listing` or `Graphic` output.
- `first=` specifies the number of top lines to display in a `Datastep` environment.
- `last=` specifies the number of bottom lines to display in a `Datastep` environment.
- `fontsize=` specifies the 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.

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

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

```
34 \define@cmdkeys{SR}[SR@]{style,dpi,firstobj,lastobj,objects,pattern}{}
35 \define@cmdkeys{SR}[SR@]{options,height,pagesize,type}{}
```

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

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

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

```
\presetkeys{SR}{%
  display  = false,   program  = false,   continued = false,
  caption  = \@empty, first    = 0,       last      = 0,
  fontsize = \@empty, scale    = \@empty, label     = \@empty,
  %
  store    = \@empty, linesize = \@empty, width     = \@empty,
  %
  type     = \@empty, style    = \@empty, dpi       = \@empty,
  firstobj = \@empty, lastobj  = \@empty, objects   = \@empty,
  pattern    = \@empty, options  = \@empty, height    = \@empty,
  pagesize = \@empty}{}
```

3.3 Customizable Settings

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

`\SRmacropath` `\SRmacropath` specifies the path to the location of the SAS macros that are bundled with the StatRep package. You can define the path by using forward slashes (/) instead of backslashes (\) as the directory name delimiter. If you do use backslashes, you must insert a backslash character into this argument by

using the \@backslashchar command. The default value is the current directory. That is, the definition for the path to the macro file is the filename itself, statrep_macros.sas.

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

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

\SRlinesize
\SRpagesize
\SRdpi
\SRstyle
\SRodsgraphopts

```
\def\SRlinesize{80}
\def\SRpagesize{500}
\def\SRdpi{300}
\def\SRstyle{statistical}
\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.

```
\def\SRgraphicdir{png}
\def\SRgraphtype{png}
\def\SRlistingdir{lst}
```

\SRgraphicdir
\SRgraphtype
\SRlistingdir

- \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.

```
\def\SRprogramname{\jobname_SR.sas}
\def\SRpreamblename{\jobname_SR_preamble.sas}
```

```
\def\SRmacroinclude{\@percentchar include "\SRmacropath" /nosource;}
\def\SRsasprogramline{\@percentchar hostdel;}%
```

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

```
%hostdel
```

```
\def\SRparindent{3em}
\def\SRintertext{... more data lines ...}
\def\SRtempfilename{sr.tmp}
\def\SRcontinuedname{continued}
\def\SRcaptionfont{\sffamily}
\def\SRcaptioncontinuedfont{\sffamily\itshape}
\def\SRverbfont{\ttfamily\bfseries}
```

\SRparindent
\SRintertext
\tempfilename

- `\SRparindent` specifies the amount of space to indent `Datastep` and `Sascode` environments. The argument is a dimension. The default is `3em` and is measured according to the font currently in use.
- `\SRintertext` specifies the text to insert in abbreviated `Datastep` environments (that is, `Datastep` environments that specify the `first=` option).
- `\SRtempfilename` specifies the name of a temporary file that is used as a scratch file in the current working directory. The default is `sr.tmp`.

\SRcontinuedname
\SRcaptionfont
\SRcaptioncontinuedfont
\SRverbfont

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

- `\SRcaptionfont` specifies the font to use for the output captions. The default is `\sffamily` (sans serif).
- `\SRcaptioncontinuedfont` specifies the font to use for the `continued` name for outputs that break across pages. The default is `\sffamily\itshape` (*sans serif, italic*).
- `\SRverbfont` specifies the font to use for code within `Datastep` and `Sascode` blocks. The default is `\ttfamily\bfseries` (typewriter text, bold).

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

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

Two output streams are created:

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

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

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.

```
\def\SR@write@preamble{
  \immediate\openout\SR@tempfile@stream\SRpreamblename%
  \SR@writepgm\SR@tempfile@stream{\SR@preambletext}
  \SR@writepgm\SR@tempfile@stream{/* Set and invoke macro variable defaults. */}
  \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultlinesize=\SRlinesize;}
  \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultpagesize=\SRpagesize;}
  \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultdpi=\SRdpi;}
  \SR@writepgm\SR@tempfile@stream{\@percentchar let defaultstyle=\SRstyle;}
  \SR@writepgm\SR@tempfile@stream{\@percentchar let listingdir=\SRlistingdir;}
  \SR@writepgm\SR@tempfile@stream{\@percentchar let graphicdir=\SRgraphicdir;}
  \SR@writepgm\SR@tempfile@stream{\@percentchar let graphtype=\SRgraphtype;}
  \SR@writepgm\SR@tempfile@stream{\@percentchar let odsgraphopts=\SRodsgraphopts;}
  \SR@writepgm\SR@tempfile@stream{}
  \SR@writepgm\SR@tempfile@stream{options nodate nonumber}
  \SR@writepgm\SR@tempfile@stream{ls=&defaultlinesize ps=&defaultpagesize}
  \SR@writepgm\SR@tempfile@stream{formchar='|----|+|---+=|-/\@backslashchar<>*';}
  \SR@writepgm\SR@tempfile@stream{}
  \SR@writepgm\SR@tempfile@stream{/* Include SAS macro definitions. */}
  \SR@writepgm\SR@tempfile@stream{\SRmacroinclude}
  \SR@writepgm\SR@tempfile@stream{}
  \immediate\closeout\SR@tempfile@stream
```

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

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

```
  \immediate\openout\SR@program@stream\SRprogramname
  \SR@writepgm\SR@program@stream{\SR@preambletext}
  \SR@writepgm\SR@program@stream{\@percentchar include "\SRpreamblename" /nosource;}
  \SR@writepgm\SR@program@stream{/* Remove all output files. */}
  \SR@writepgm\SR@program@stream{\SRsasprogramline}
  \SR@writepgm\SR@program@stream{}
  \SR@writepgm\SR@program@stream{/* Start program with a null title. */}
  \SR@writepgm\SR@program@stream{title;}
  \SR@writepgm\SR@program@stream{}
}
```

`\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.

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

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

3.4 Code Environments

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

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

- \verbatim@addtoline adds its argument to the character buffer.
- \verbatim@processline typesets the characters that accumulate in the buffer.

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

3.4.1 Datastep Environment

Three macros are defined to set up the Datastep environment:

```
\def\SR@datastep@writefile#1{%
  \setcounter{SR@totallines}{0}%
  \@bsphack\immediate\openout\SR@tempfile@stream#1%
  \let\do\@makeother\dospecials\catcode`\^^M\active%
  %
  \def\verbatim@processline{%
```

```
    \addtocounter{SR@totallines}{1}%
    \SR@writepgm\SR@tempfile@stream{\the\verbatim@line}%
  }%
  \verbatim@start%
}
```

\SR@datastep@writefile

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

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

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

\SR@enddatastep@writefile

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

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

\SR@datastep@writeprogramline

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

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

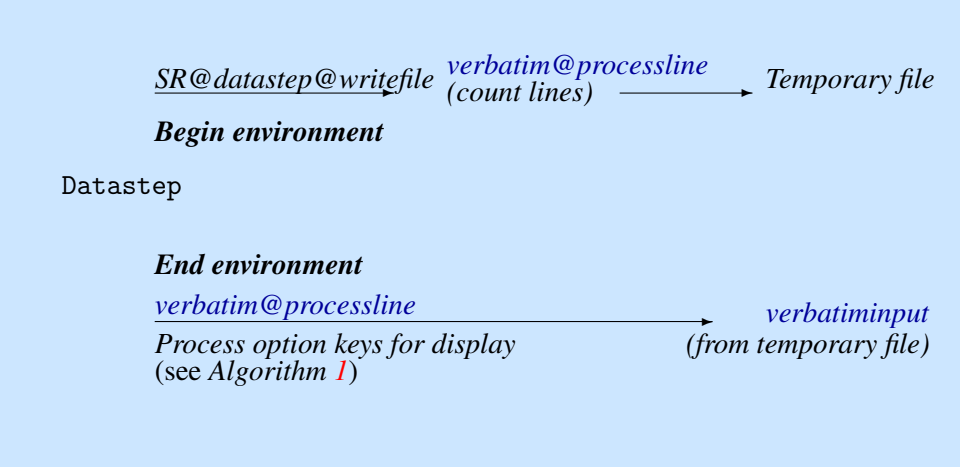

Figure 2: Processing Schematic for the Datastep Environment

```
\def\@Datastep[#1]{%
  \setkeys{SR}{#1}%
  \SR@datastep@writefile{\SRtempfilename}%
}
\newenvironment{Datastep}{%
  \catcode`\^^M=\active
  \@ifnextchar[{\catcode`\^^M=5\@Datastep}{\catcode`\^^M=5\@Datastep[]}
  }
```

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

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

```
{%
```

```
\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.

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

Otherwise, the code block is to be displayed.

```
    \ifthenelse{\equal{\SR@fontsize}{\@empty}}%
    {\relax}%
    {\@nameuse{\SR@fontsize}}%
    \setcounter{SR@currentline}{0}%
    \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.

```
    \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.

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

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

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

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

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

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

```
152       \ifnum\SR@first>0%
153         \ifnum\theSR@currentline<\SR@first%
154           {\hskip\SRparindent\the\verbatim@line\par}%
155         \else%
156           \ifnum\theSR@currentline=\SR@first%
157             \par\hskip\SRparindent\SRintertext\par%
158           \else%
159             \ifnum\SR@last>0%
160               \ifnum\theSR@currentline=\theSR@startinglastline\par\fi%
161               \ifnum\theSR@currentline>\theSR@startinglastline%
162                 {\hskip\SRparindent\the\verbatim@line\par}%
163               \fi%
164             \fi%
165           \fi%
166         \fi%
167       \else%
168         \hskip\SRparindent\the\verbatim@line\par%
169       \fi%
170       \addtocounter{SR@currentline}{1}%
171     }%
172   \fi%
```

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

macro processes each line.

```
for each line in block do
    if block is not display only then
        write data line ;
    if first=n is specified then
        if currentline < n then
            display line;
        else if currentline = n then
            write the value of \SRintertext;
        else if last=m is specified then
            if currentline = (totallines - m) then
                start new line;
            else if currentline > (totallines - m) then
                display line;
    else
        display line;
end
```

Algorithm 1: `Datastep` Environment: `\verbatim@processline` Macro

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

```
\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:

```
\ifSR@program
 \setlength{\SR@scratchlength}{-2\topsep-\partopsep-2\parskip-2\baselineskip}
 \vspace*{\SR@scratchlength}%
\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:

```
\ifSR@display\else%
  \SR@writepgm\SR@program@stream{}%
\fi%
}
```

3.4.2 Sascode Environment

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

- \SR@sascode@addtoline resets category codes and calls \SR@sascode@filter.
- \SR@sascode@filter acts as a filter. It processes lines with line commands and passes other lines to \SR@sascode@writeline.
- \SR@sascode@writeline writes the lines into the character buffer that is used by the verbatim package. It also writes lines to the generated program file.

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

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

```
\begingroup
  \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.

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

After expansion, the following command is left, where *n* is the original category code of the % character:
`\relax\SR@sascode@filter` *n*
That is, the macro first redefines the category code of the % character and calls the `\SR@sascode@filter` macro with the original category code of the % character as its first argument.

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

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

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

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

The regular expression argument must be escaped for use in LaTeX. For example, consider the task to capture the integer in the following expression:
`%* program 4;`
A typical regular expression to capture the integer 4 is defined as follows:
`^%\*\s*program\s*([0-9]+);`

To use the expression in LaTeX, it must be escaped so that the pattern becomes
`^%\string\*\space*program\space*([0-9]+);`

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

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

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

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

```
for each line in block do
    if line begins with %* (column 1) then
        if line matches %* program n;
        then
            set program counter
        else if line matches %* display n;
        then
            set display counter
        else if line matches %* ;
        then
            write line to program;
        else
            display and write to program (valid SAS macro comment)
        end
    else
        pass line onward to \SR@sascode@writeline;
    end
end
```

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

```
\gdef\SR@sascode@filter#1#2{
  \ifnum\pdfmatch{^%\string\*}
                 {\detokenize{#2}}=1
    \ifnum\pdfmatch{^%\string\*\space*program\space*([0-9]+);}
                   {\detokenize{#2}}=1
      \setcounter{SR@programline}
                 {\expandafter\strip@prefix\pdflastmatch 1}
      \catcode`\%#1\@tempswafalse\verbatim@line{}
    \else
      \ifnum\pdfmatch{^%\string\*\space*display\space*([0-9]+);}
                     {\detokenize{#2}}=1
        \setcounter{SR@displayline}
                   {\expandafter\strip@prefix\pdflastmatch 1}
        \catcode`\%#1\@tempswafalse\verbatim@line{}
      \else
        \ifnum\pdfmatch{^%\string\*\space*;(.*)$}
                       {\detokenize{#2}}=1
          \SR@writepgm\SR@program@stream{\expandafter\strip@prefix\pdflastmatch 1}
          \catcode`\%#1\@tempswafalse\verbatim@line{}
```

```
          \else \catcode`\%#1\SR@sascode@writeline{#2}
          \fi
        \fi
      \fi
    \else
      \catcode`\%#1\SR@sascode@writeline{#2}
    \fi
  }
\endgroup
```

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

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

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

The \SR@sascode@filter macro passes all lines that are not line commands to the \SR@sascode@writeline macro. Lines that begin with the special line 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.

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

```
    \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
    \@tempswafalse%
  \else%
    \ifSR@display%
      \verbatim@line\expandafter{\the\verbatim@line\hskip\SRparindent#1}%
    \else%
      \ifnum\theSR@programline>0%
        \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
        \addtocounter{SR@programline}{-1}%
        \@tempswafalse%
      \else%
        \ifnum\theSR@displayline>0%
          \verbatim@line\expandafter{\the\verbatim@line\hskip\SRparindent#1}%
          \addtocounter{SR@displayline}{-1}%
        \else%
          \SR@writepgm\expandafter\SR@program@stream{\detokenize{#1}}%
          \verbatim@line\expandafter{\the\verbatim@line\hskip\SRparindent#1}%
        \fi%
      \fi%
    \fi%
  \fi%
}
```

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

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

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

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

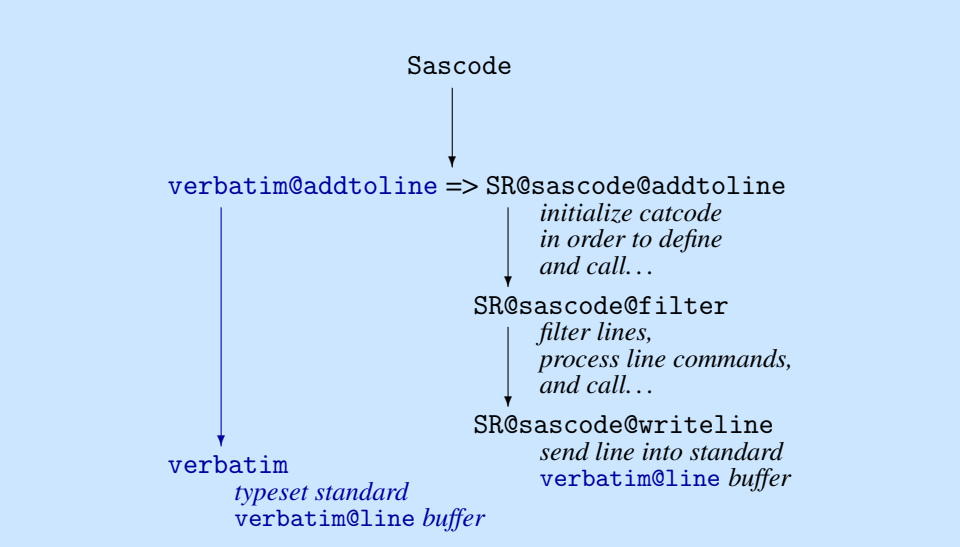

Figure 3: Processing Schematic for the Sascode Environment

Sascode

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

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

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

1. The verbatim package macro calls \verbatim@addtoline.
2. The \verbatim@addtoline macro redirects to \SR@sascode@addtoline.
3. The \SR@sascode@addtoline macro sets category codes and calls \SR@sascode@filter.
4. The \SR@sascode@filter macro processes line commands and calls \SR@sascode@writeline.
5. The \SR@sascode@writeline macro parses and processes the lines. It writes the appropriate lines to the generated program file. It also writes the appropriate lines into the \verbatim@line buffer, which is processed by

the `verbatim` package macro `\verbatim`.

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

```
\def\@Sascode[#1]{%
  \setkeys{SR}{#1}%
  \ifSR@display\else%
    \ifthenelse{\equal{\SR@store}{\@empty}}%
      {\relax}%
      {\SR@writepgm\SR@program@stream{\@percentchar output(\SR@store)}%
       \PackageInfo{statrep}{Processing Sascode block \SR@store}}%
  \fi%
  \ifthenelse{\equal{\SR@fontsize}{\@empty}}%
    {\relax}%
    {\@nameuse{\SR@fontsize}}%
  \setcounter{SR@programline}{0}%
  \setcounter{SR@displayline}{0}%
  \let\verbatim@addtoline\SR@sascode@addtoline%
  \verbatim}
\newenvironment{Sascode}{%
  \catcode`\^^M=\active
  \@ifnextchar[{\catcode`\^^M=5\@Sascode}{\catcode`\^^M=5\@Sascode[]}
}
```

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:

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

```
%
\ifSR@display\else%
  \ifthenelse{\equal{\SR@store}{\@empty}}%
    {\relax}%
    {\SR@writepgm\SR@program@stream{\@percentchar endoutput(\SR@store)}%
     \SR@writepgm\SR@program@stream{}}%
\fi%
}
```

3.5 Handling the SAS Output

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

Graphical output is inserted as specified in the `\Graphic` tag, captioned, and centered. Listing output is inserted in the same way, but the output is framed. The frame is created with the help of the `longfigure` package and the `\hline` command.

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

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

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

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

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

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

```
\def\SR@write@outoptions{
```

```
    \SR@set@outoptions
    \let\#\SR@hashchar
    \SR@writepgm\SR@program@stream{%
      \@percentchar write(\SR@label,store=\SR@store \SR@final@outoptions)
    }
```

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

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

`\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.

```
\newcommand*{\SR@set@outoptions}{%
  \global\edef\SR@final@outoptions{}
  \SR@outkeyval{height}\SR@outkeyval{objects}\SR@outkeyval{pattern}
  \SR@outkeyval{style}\SR@outkeyval{dpi}\SR@outkeyval{options}
  \SR@outkeyval{pagesize}\SR@outkeyval{linesize}\SR@outkeyval{width}
  \SR@outkeyval{type}\SR@outkeyval{firstobj}\SR@outkeyval{lastobj}
  }
```

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.

```
\newcommand*{\SR@outkeyval}[1]{%
  \def\SR@outkey{#1}
  \edef\SR@outval{\csname SR@#1\endcsname}
  \ifx\@empty\SR@outval
  \else
    \edef\SR@final@outoptions{\SR@final@outoptions,\SR@outkey=\SR@outval}
  \fi
  }
```

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

\Listing inserts a stream of listing outputs.

\Boxlisting boxes a single listing.

\Graphic inserts a stream of graphical outputs.

\Boxgraphic boxes a single graphic.

\SR@insert begins the output insertion process.

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

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

3.5.1 The \Listing Macro

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

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

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

The following statements set the font size:

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

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

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

The following statements control the line size:

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

```
306   \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]

```
307   \setlength{\SR@scratchlength}{\SR@verbwidth+2\tabcolsep+
308                                                 2\arrayrulewidth}%
```

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

```
309   \SR@insert{#2}%
310 }
```

3.5.2 The \Graphic Macro

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

```
311 \newcommand{\Graphic}[2][]{%
312   \setkeys{SR}{#1,type=graphic, label=#2}%
313   \ifthenelse{\equal{\SR@store}{\@empty}}
314     {\relax}
315     {\SR@write@outoptions}
316   \SR@insert{#2}%
```

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

```
}
```

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

Finally, \SR@insert is called to insert the \Graphic.

3.5.3 The \Boxlisting Macro

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

```
\newcommand*{\Boxlisting}[1]{%
  \global\def\SR@insertname{\SRlistingdir/#1.lst}%
  \IfFileExists{\SR@insertname}%
    {%
      \savebox{\SR@filebox}{%
          \def\verbatim@font{\@nameuse{\SR@fontsize}\SRverbfont}%
          \parbox{\SR@verbwidth}{%
            \vspace*{0.25\baselineskip}%
            \noindent\verbatiminput{\SR@insertname}%
            \vspace*{0.25\baselineskip}}
          }%
    }%
    {\global\def\SR@insertname{\@empty}%
     \savebox{\SR@filebox}{\fbox{Missing File}}%
    }%
  \global\setbox\SR@filebox=\box\SR@filebox%
  \PackageInfo{statrep}{Processing listing \SR@insertname}
}
```

\Boxlisting The \Boxlisting macro takes a single argument, the {⟨*filename*⟩} of the listing output to be boxed (the filename without the extension). If the file exists, the macro places the content of the file (using \verbatiminput) into a box of \SR@verbwidth, which is calculated as described in section 3.5.1. White space at the top and bottom of the box is inserted (0.25\baselineskip).

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

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

3.5.4 The \Boxgraphic Macro

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

```
\newcommand*{\Boxgraphic}[1]{%
  \IfFileExists{\SRgraphicdir/#1.\SRgraphtype}%
    {\global\def\SR@insertname{\SRgraphicdir/#1.\SRgraphtype}}
    {\global\def\SR@insertname{\@empty}}%
  \ifthenelse{\equal{\SR@insertname}{\@empty}}%
    {\savebox{\SR@filebox}{\fbox{ Missing File \SRgraphicdir/#1.\SRgraphtype}}}%
    {%
      \ifthenelse{\equal{\SR@scale}{\@empty}}%
        {\ifthenelse{\equal{\SR@width}{\@empty}}%
          {\savebox{\SR@filebox}{\includegraphics{\SR@insertname}}}%
          {\savebox{\SR@filebox}{\includegraphics[width=\SR@width]%
                                                 {\SR@insertname}}}}%
        {\savebox{\SR@filebox}{\includegraphics[scale=\SR@scale]%
                                               {\SR@insertname}}}%
    }%
  \global\setbox\SR@filebox=\box\SR@filebox%
  \PackageInfo{statrep}{Processing graphic \SR@insertname}%
  \setlength{\SR@scratchlength}{\wd\SR@filebox+2\tabcolsep}%
}%
```

\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 PNG file exists. If it does, it sets the global variable \SR@insertname to the full name. Otherwise, it checks whether the PDF file exists and if that file exists, it sets the global variable to the full name.

If the file does not exist in either PNG or PDF format, \SR@insertname is defined to be \@empty.

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

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

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

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

3.5.5 The \SR@insert Macro

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

SR@filebox

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

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

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

\SR@set@outheadings

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

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

The following statements define three macros for typing shortcuts:

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

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

```
  \ifthenelse{\equal{\SR@type}{listing}}
             {\def\SR@@firsthead{\hline\endLFfirsthead}
              \def\SR@@conthead{\hline\endLFhead}
              \def\SR@@endfoot{\hline\endLFfoot}}
             %
             {\def\SR@@firsthead{\endLFfirsthead}
              \def\SR@@conthead{\endLFhead}
              \def\SR@@endfoot{\endLFfoot}}
  %%%
```

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

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

```
  %%%
\renewcommand*{\SR@firsthead}{\SR@@firsthead}%
\renewcommand*{\SR@conthead}{\SR@@conthead}%
\renewcommand*{\SR@endfoot}{\SR@@endfoot}%

 \ifthenelse{\equal{\SR@caption}{\@empty}}%
   {% CAPTION NOT GIVEN
    \renewcommand*{\SR@firsthead}{}%
    \ifSR@continued%
    \addtocounter{\LFcounter}{-1}
      \renewcommand*{\SR@conthead}{\SR@@captioncontinued%
                     \SR@@captionskip\SR@@conthead}
   \fi%
   }%
```

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

```
   {% CAPTION GIVEN
   \renewcommand*{\SR@firsthead}{\SR@@caption\label{\SR@label}%
                                  \SR@@captionskip\SR@@firsthead}%
   \renewcommand*{\SR@conthead}{\SR@@captioncontinued%
                                 \SR@@captionskip\SR@@conthead}
   }
}
```

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

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

```
\def\SR@set@outmargin{%
   \ifdim\SR@scratchlength>\textwidth%
```

```
      \dosloppy%
      \setlength{\LFleft}{(\paperwidth-\SR@scratchlength)/\real{2.0}}%
      \addtolength{\LFleft}{-1in}%
      \addtolength{\LFleft}{-\hoffset}%
      \addtolength{\LFleft}{-\leftskip}%
      \ifodd\c@page\addtolength{\LFleft}{-\oddsidemargin}%
      \else\addtolength{\LFleft}{-\evensidemargin}%
      \fi
    \else
      \setlength{\LFleft}{(\textwidth-\SR@scratchlength)/\real{2.0}}%
    \fi
}
```

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

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

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

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

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

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

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

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:

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

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

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

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

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

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

```
\newcommand{\SR@insert}[1]{%
  \SR@set@outheadings%
  \ifthenelse{\equal{\SR@type}{listing}}{\Boxlisting{#1}}{\Boxgraphic{#1}}
  \SR@set@outmargin%
```

The width of a \Listing output is calculated in the \Listing macro. The width of a \Graphic output is calculated in the \Boxgraphic macro. Therefore, after boxing an output, the \SR@scratchlength contains the width of the output regardless of type. With that information, the margins that are required for centering are set.

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

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

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

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

```
413         \setcounter{SR@multifilecount}{1}%
414         \whiledo{\boolean{SR@multifile}}{%
415           \ifthenelse{\equal{\SR@type}{listing}}
416             {\Boxlisting{#1\arabic{SR@multifilecount}}}%
417             {\Boxgraphic{#1\arabic{SR@multifilecount}}}%
418           \ifthenelse{\equal{\SR@insertname}{\@empty}}%
419             {\setboolean{SR@multifile}{false}}%
420             {\\[0.5\baselineskip]\usebox{\SR@filebox}%
421           \stepcounter{SR@multifilecount}%
422           \setboolean{SR@multifile}{true}}%
423         }%
424       \endlongfigure%
425     \unsloppy%
426 }
```

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

```
while multiple files is True do
    if type=listing then
        Boxlisting
    else
        Boxgraphic
    end
    if file exists then
        insert contents, increment counter
    else
        multiple files is False
    end
end
```

Algorithm 4: Input Loop for the Output Stream

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

When no file matches the pattern (the `\SR@insertname` macro is `\@empty`), the

Boolean switch \SR@multifile is set to false and the loop is ended.

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

<*longfigure>

4 longfigure Usage

The longfigure package differs slightly from the well-known longtable package written by David Carlisle. The longtable package defines a longtable environment, which produces tables that can be broken by TEX's standard page-breaking algorithm. Similarly, the longfigure package defines a longfigure environment that produces figures that can be broken by TEX's standard page-breaking algorithm.

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

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

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

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

```
\listoffigures
```

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

```
\listofdisplay
```

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

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

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

5 longfigure Implementation

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

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

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

```
428 \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.

```
\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.

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

5.1 Options

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

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

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

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

The following statements process the options:

```
\def\LFProcessOptions#1{
  \@ifundefined{c@#1}{%
      \RequirePackage{tocloft}
      \def\LFuc##1##2{\MakeUppercase{##1}{##2}}
      \expandafter\def\csname list#1name\endcsname{List of \LFuc#1s}
      \ifx\@empty\LFreset%
        \newlistof{#1}{lft}{\csname list#1name\endcsname}
      \else
        \newlistof[\LFreset]{#1}{lft}{\csname list#1name\endcsname}
      \fi
  }{}%
}
\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.

```
\newif\ifLF@same
\newcommand{\strcfstr}[2]{%
  \LF@samefalse
  \begingroup\def\2{#2}
    \ifx\2#1\endgroup\LF@sametrue
    \else\endgroup
    \fi
}
```

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

```
\def\LFupcase#1{%
  \def\x##1##2{%
    \MakeUppercase{##1}{##2}}\x#1%
}
```

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).

```
\def\LF@name{\expandafter\LFupcase%
```

```
              \expandafter{\LFcounter}~%
              \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.

```
\def\LF@err{\PackageError{longfigure}}
\def\LF@warn{\PackageWarning{longfigure}}
\def\LF@final@warn{%
  \AtEndDocument{%
    \LF@warn{\LFcounter \@width s have changed. Rerun \LaTeX\.\@gobbletwo}}%
  \global\let\LF@final@warn\relax}
%
\newskip\LFleft       \LFleft=\fill
\newskip\LFright      \LFright=\fill
\newskip\LFpre        \LFpre=\bigskipamount
\newskip\LFpost       \LFpost=\bigskipamount
\newcount\LFchunksize \LFchunksize=20
\let\c@LFchunksize\LFchunksize
\newdimen\LFcapwidth  \LFcapwidth=4in
\newbox\LF@head
\newbox\LF@firsthead
\newbox\LF@foot
\newbox\LF@lastfoot
\newcount\LF@cols
\newcount\LF@rows
\newcounter{LF@tables}
\newcounter{LF@chunks}[LF@tables]
%
\newtoks\LF@p@ftn
\mathchardef\LF@end@pen=30000
\def\longfigure{%
  \par
  \ifx\multicols\@undefined
  \else
     \ifnum\col@number>\@ne
       \@twocolumntrue
     \fi
  \fi
  \if@twocolumn
    \LF@err{longfigure not in 1-column mode}\@ehc
  \fi
  \begingroup
  \@ifnextchar[\LF@array{\LF@array[x]}}
```

```
\def\LF@array[#1]#2{%
  \refstepcounter{\LFcounter}\stepcounter{LF@tables}%
  \if l#1%
    \LFleft\z@ \LFright\fill
  \else\if r#1%
    \LFleft\fill \LFright\z@
  \else\if c#1%
    \LFleft\fill \LFright\fill
  \fi\fi\fi
  \let\LF@mcol\multicolumn
  \let\LF@@tabarray\@tabarray
  \let\LF@@hl\hline
  \def\@tabarray{%
    \let\hline\LF@@hl
    \LF@@tabarray}%
  \let\\\LF@tabularcr\let\tabularnewline\\%
  \def\newpage{\noalign{\break}}%
  \def\pagebreak{\noalign{\ifnum`}=0\fi\@testopt{\LF@no@pgbk-}4}%
  \def\nopagebreak{\noalign{\ifnum`}=0\fi\@testopt\LF@no@pgbk4}%
  \let\hline\LF@hline \let\kill\LF@kill\let\caption\LF@caption
  \@tempdima\ht\strutbox
  \let\@endpbox\LF@endpbox
  \ifx\extrarowheight\@undefined
    \let\@acol\@tabacol
    \let\@classz\@tabclassz \let\@classiv\@tabclassiv
    \def\@startpbox{\vtop\LF@startpbox}%
    \let\@@startpbox\@startpbox
    \let\@@endpbox\@endpbox
    \let\LF@LL@FM@cr\@tabularcr
  \else
    \advance\@tempdima\extrarowheight
    \col@sep\tabcolsep
    \let\@startpbox\LF@startpbox\let\LF@LL@FM@cr\@arraycr
  \fi
  \setbox\@arstrutbox\hbox{\vrule
    \@height \arraystretch \@tempdima
    \@depth \arraystretch \dp \strutbox
    \@width \z@}%
  \let\@sharp##\let\protect\relax
   \begingroup
    \@mkpream{#2}%
    \xdef\LF@bchunk{%
       \global\advance\c@LF@chunks\@ne
       \global\LF@rows\z@\setbox\z@\vbox\bgroup
       \LF@setprevdepth
```

```
      \tabskip\LFleft \noexpand\halign to\hsize\bgroup
     \tabskip\z@ \@arstrut \@preamble \tabskip\LFright \cr}%
  \endgroup
  \expandafter\LF@nofcols\LF@bchunk&\LF@nofcols
  \LF@make@row
  \m@th\let\par\@empty
  \everycr{}\lineskip\z@\baselineskip\z@
  \LF@bchunk}
\def\LF@no@pgbk#1[#2]{\penalty #1\@getpen{#2}\ifnum‘{=0\fi}}
\def\LF@start{%
  \let\LF@start\endgraf
  \endgraf\penalty\z@\vskip\LFpre
  \dimen@\pagetotal
  \advance\dimen@ \ht\ifvoid\LF@firsthead\LF@head\else\LF@firsthead\fi
  \advance\dimen@ \dp\ifvoid\LF@firsthead\LF@head\else\LF@firsthead\fi
  \advance\dimen@ \ht\LF@foot
  \dimen@ii\vfuzz
  \vfuzz\maxdimen
    \setbox\tw@\copy\z@
    \setbox\tw@\vsplit\tw@ to \ht\@arstrutbox
    \setbox\tw@\vbox{\unvbox\tw@}%
  \vfuzz\dimen@ii
  \advance\dimen@ \ht
        \ifdim\ht\@arstrutbox>\ht\tw@\@arstrutbox\else\tw@\fi
  \advance\dimen@\dp
        \ifdim\dp\@arstrutbox>\dp\tw@\@arstrutbox\else\tw@\fi
  \advance\dimen@ -\pagegoal
  \ifdim \dimen@>\z@\vfil\break\fi
      \global\@colroom\@colht
  \ifvoid\LF@foot\else
    \advance\vsize-\ht\LF@foot
    \global\advance\@colroom-\ht\LF@foot
    \dimen@\pagegoal\advance\dimen@-\ht\LF@foot\pagegoal\dimen@
    \maxdepth\z@
  \fi
  \ifvoid\LF@firsthead\copy\LF@head\else\box\LF@firsthead\fi\nobreak
  \output{\LF@output}}
\def\endlongfigure{%
  \crcr
  \noalign{%
    \let\LF@entry\LF@entry@chop
    \xdef\LF@save@row{\LF@save@row}}%
  \LF@echunk
  \LF@start
  \unvbox\z@
```

```
  \LF@get@widths
  \if@filesw
    {\let\LF@entry\LF@entry@write\immediate\write\@auxout{%
      \gdef\expandafter\noexpand
        \csname LF@\romannumeral\c@LF@tables\endcsname
          {\LF@save@row}}}%
  \fi
  \ifx\LF@save@row\LF@@save@row
  \else
    \LF@warn{Column \@width s have changed\MessageBreak
             in table \thetable}%
    \LF@final@warn
  \fi
  \endgraf\penalty -\LF@end@pen
  \endgroup
  \global\@mparbottom\z@
  \pagegoal\vsize
  \endgraf\penalty\z@\addvspace\LFpost
  \ifvoid\footins\else\insert\footins{}\fi}
\def\LF@nofcols#1&{%
  \futurelet\@let@token\LF@n@fcols}
\def\LF@n@fcols{%
  \advance\LF@cols\@ne
  \ifx\@let@token\LF@nofcols
    \expandafter\@gobble
  \else
    \expandafter\LF@nofcols
  \fi}
\def\LF@tabularcr{%
  \relax\iffalse{\fi\ifnum0=`}\fi
  \@ifstar
    {\def\crcr{\LF@crcr\noalign{\nobreak}}\let\cr\crcr
     \LF@t@bularcr}%
    {\LF@t@bularcr}}
\let\LF@crcr\crcr
\let\LF@setprevdepth\relax
\def\LF@t@bularcr{%
  \global\advance\LF@rows\@ne
  \ifnum\LF@rows=\LFchunksize
    \gdef\LF@setprevdepth{%
      \prevdepth\z@\global
      \global\let\LF@setprevdepth\relax}%
    \expandafter\LF@xtabularcr
  \else
    \ifnum0=`{}\fi
```

```
    \expandafter\LF@LL@FM@cr
  \fi}
\def\LF@xtabularcr{%
  \@ifnextchar[\LF@argtabularcr\LF@ntabularcr}
\def\LF@ntabularcr{%
  \ifnum0=`{}\fi
  \LF@echunk
  \LF@start
  \unvbox\z@
  \LF@get@widths
  \LF@bchunk}
\def\LF@argtabularcr[#1]{%
  \ifnum0=`{}\fi
  \ifdim #1>\z@
    \unskip\@xargarraycr{#1}%
  \else
    \@yargarraycr{#1}%
  \fi
  \LF@echunk
  \LF@start
  \unvbox\z@
  \LF@get@widths
  \LF@bchunk}
\def\LF@echunk{%
  \crcr\LF@save@row\cr\egroup
  \global\setbox\@ne\lastbox
    \unskip
  \egroup}
\def\LF@entry#1#2{%
  \ifhmode\@firstofone{&}\fi\omit
  \ifnum#1=\c@LF@chunks
  \else
    \kern#2\relax
  \fi}
\def\LF@entry@chop#1#2{%
  \noexpand\LF@entry
    {\ifnum#1>\c@LF@chunks
       1}{0pt%
     \else
       #1}{#2%
     \fi}}
\def\LF@entry@write{%
  \noexpand\LF@entry^^J%
  \@spaces}
\def\LF@kill{%
```

```
  \LF@echunk
  \LF@get@widths
  \expandafter\LF@rebox\LF@bchunk}
\def\LF@rebox#1\bgroup{%
  #1\bgroup
  \unvbox\z@
  \unskip
  \setbox\z@\lastbox}
\def\LF@blank@row{%
  \xdef\LF@save@row{\expandafter\LF@build@blank
    \romannumeral\number\LF@cols 001 }}
\def\LF@build@blank#1{%
  \if#1m%
    \noexpand\LF@entry{1}{0pt}%
    \expandafter\LF@build@blank
  \fi}
\def\LF@make@row{%
  \global\expandafter\let\expandafter\LF@save@row
    \csname LF@\romannumeral\c@LF@tables\endcsname
  \ifx\LF@save@row\relax
    \LF@blank@row
  \else
    {\let\LF@entry\or
     \if!%
         \ifcase\expandafter\expandafter\expandafter\LF@cols
         \expandafter\@gobble\LF@save@row
         \or
         \else
           \relax
         \fi
        !%
     \else
       \aftergroup\LF@blank@row
     \fi}%
  \fi}
\let\setlongfigures\relax
\def\LF@get@widths{%
  \setbox\tw@\hbox{%
    \unhbox\@ne
    \let\LF@old@row\LF@save@row
    \global\let\LF@save@row\@empty
    \count@\LF@cols
    \loop
      \unskip
      \setbox\tw@\lastbox
```

```
    \ifhbox\tw@
      \LF@def@row
      \advance\count@\m@ne
    \repeat}%
  \ifx\LF@@save@row\@undefined
    \let\LF@@save@row\LF@save@row
  \fi}
\def\LF@def@row{%
  \let\LF@entry\or
  \edef\@tempa{%
    \ifcase\expandafter\count@\LF@old@row
    \else
      {1}{0pt}%
    \fi}%
  \let\LF@entry\relax
  \xdef\LF@save@row{%
    \LF@entry
    \expandafter\LF@max@sel\@tempa
    \LF@save@row}}
\def\LF@max@sel#1#2{%
  {\ifdim#2=\wd\tw@
     #1%
   \else
     \number\c@LF@chunks
   \fi}%
  {\the\wd\tw@}}
\def\LF@hline{%
  \noalign{\ifnum0=`}\fi
    \penalty\@M
    \futurelet\@let@token\LF@@hline}
\def\LF@@hline{%
  \ifx\@let@token\hline
    \global\let\@gtempa\@gobble
    \gdef\LF@sep{\penalty-\@medpenalty\vskip\doublerulesep}%
  \else
    \global\let\@gtempa\@empty
    \gdef\LF@sep{\penalty-\@lowpenalty\vskip-\arrayrulewidth}%
  \fi
  \ifnum0=`{\fi}%
  \multispan\LF@cols
     \unskip\leaders\hrule\@height\arrayrulewidth\hfill\cr
  \noalign{\LF@sep}%
  \multispan\LF@cols
     \unskip\leaders\hrule\@height\arrayrulewidth\hfill\cr
  \noalign{\penalty\@M}%
```

```
\@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`:

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

The `\LF@caption` 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`.

```
\def\LF@c@ption#1[#2]#3{%
  \LF@makecaption#1\LF@name{#3}%
  \def\@tempa{#2}%
  \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.

```
    {\let\\\space
    \strcfstr{\LFcounter}{figure}
      \ifLF@same\def\LFoutfile{lof}\else
        \strcfstr{\LFcounter}{table}
          \ifLF@same\def\LFoutfile{lot}\else
            \def\LFoutfile{lft}\fi\fi
    \addcontentsline{\LFoutfile}{\LFcounter}
    {\expandafter\protect\expandafter\numberline\expandafter%
  {\expandafter\csname the\LFcounter\endcsname}{#2}}}%
  \fi
}
```

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

```
\def\LF@capti@n{%
  \@ifstar
    {\egroup\LF@c@ption\@gobble[]}%
    {\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.

```
\def\LF@makecaption#1#2#3{%
  \LF@mcol\LF@cols c{\hbox to\z@{\hss\parbox[t]\LFcapwidth{%
    \sbox\@tempboxa{#1{#2: }#3}%
    \ifdim\wd\@tempboxa>\hsize
      #1{#2: }#3%
    \else
      \hbox to\hsize{\hfil\box\@tempboxa\hfil}%
    \fi
    \endgraf\vskip\baselineskip}%
  \hss}}}
\def\LF@output{%
  \ifnum\outputpenalty <-\@Mi
    \ifnum\outputpenalty > -\LF@end@pen
      \LF@err{floats and marginpars not allowed in a longfigure}\@ehc
    \else
      \setbox\z@\vbox{\unvbox\@cclv}%
      \ifdim \ht\LF@lastfoot>\ht\LF@foot
        \dimen@\pagegoal
        \advance\dimen@-\ht\LF@lastfoot
        \ifdim\dimen@<\ht\z@
          \setbox\@cclv\vbox{\unvbox\z@\copy\LF@foot\vss}%
          \@makecol
          \@outputpage
          \setbox\z@\vbox{\box\LF@head}%
        \fi
      \fi
      \global\@colroom\@colht
      \global\vsize\@colht
      \vbox
        {\unvbox\z@\box\ifvoid\LF@lastfoot\LF@foot\else
         \LF@lastfoot\fi}%
    \fi
```

```
  \else
    \setbox\@cclv\vbox{\unvbox\@cclv\copy\LF@foot\vss}%
    \@makecol
    \@outputpage
      \global\vsize\@colroom
    \copy\LF@head\nobreak
  \fi}
\def\LF@end@hd@ft#1{%
  \LF@echunk
  \ifx\LF@start\endgraf
    \LF@err
     {Longfigure head or foot not at start of table}%
     {Increase LFchunksize}%
  \fi
  \setbox#1\box\z@
  \LF@get@widths
  \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;

```
\def\endLFfirsthead{\LF@end@hd@ft\LF@firsthead}
\def\endLFhead{\LF@end@hd@ft\LF@head}
\def\endLFfoot{\LF@end@hd@ft\LF@foot}
\def\endLFlastfoot{\LF@end@hd@ft\LF@lastfoot}
 %
\def\LF@startpbox#1{%
  \bgroup
    \let\@footnotetext\LF@p@ftntext
    \setlength\hsize{#1}%
    \@arrayparboxrestore
    \vrule \@height \ht\@arstrutbox \@width \z@}
\def\LF@endpbox{%
  \@finalstrut\@arstrutbox
  \egroup
  \the\LF@p@ftn
  \global\LF@p@ftn{}%
  \hfil}
\def\LF@p@ftntext#1{%
  \edef\@tempa{\the\LF@p@ftn\noexpand\footnotetext[\the\c@footnote]}%
  \global\LF@p@ftn\expandafter{\@tempa{#1}}}%
```

5.4 References

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

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

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

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

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