boxenv.sty: A LaTeX style file for formatting BOX expressions

Merijn de Jonge

Software Engineering (SEN)

SEN-R9911 May 1999

Report SEN-R9911
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

`boxenv.sty`: A LaTeX Style File for Formatting BOX Expressions

Merijn de Jonge
CWI
P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands
M.de.Jonge@cwi.nl

ABSTRACT

BOX is a language independent mark-up language. It is designed for use within a generic pretty-print framework to connect source language dependent *front-ends* to target language dependent *back-ends*. A front-end translates a term over a language to BOX to describe its intended layout. Back-ends translate BOX terms to arbitrary output formats. The tool `box2latex` is a back-end that generates LaTeX code using specialized LaTeX commands and environments defined in the style file `boxenv.sty`. This style file is required in order to process the generated document by LaTeX. This paper describes how to integrate the generated LaTeX files within your documents, and it describes the low-level interface and implementation of the `boxenv.sty` style file.

1991 Computing Reviews Classification System: D.2.1, D.2.3, D.2.6, D.2.7, D.2.m, D.3.2, I.7.2.

Keywords and Phrases: typesetting, pretty-printing, box language, software documentation, document preparation, LaTeX.

Note: Work carried out under project SEN-1.4, ASF+SDF

1 Introduction

This document describes the use and implementation of the `boxenv.sty` style file, which, in combination with `box2latex`, is used to format arbitrary BOX expressions.

The style file and the program `box2latex` are part of the generic pretty-print system described in [7]. A generic pretty-printer according to that article, consists of a front-end and a back-end which are connected by the language independent mark-up language BOX. A front-end translates a term over some language to a BOX expression which describes the intended layout of that term. A back-end translates a BOX expression to some output format. In [7] three back-ends are described: i) `box2text` which translates a BOX expression to its textual representation; ii) `box2asfix` which updates the layout in an AsFix [8] term according to a BOX term; iii) `box2latex` which translates a BOX term to LaTeX.

operator	*space options*	*description*
H	hs	Formats its sub-boxes horizontally.
V	vs, is	Formats its sub-boxes vertically.
HV	hs, vs, is	Respects line width by formatting its sub-boxes horizontally and vertically.
A	hs, vs	Formats its sub-boxes in a tabular.
ALT		Depending on the available width, formats its first or second sub-box.

Table 1: Available positional BOX operators and supported space options (hs defines horizontal layout between boxes, vs defines vertical layout between boxes, and is defines left indentation).

The `box2latex` back-end uses special LaTeX environments for its translation of BOX. For each BOX operator a corresponding LaTeX environment is used. The translation from BOX to LaTeX consists of recursively translating the BOX operators in a BOX term to their corresponding LaTeX environments. For example, text placed within the H BOX operator will be put in a `HBOX` environment by `box2latex`. The style file `boxenv.sty`, in which these environments are defined, is required in order to be able to process the generated files by LaTeX.

This document is organized as follows: First we give a brief overview of the BOX language. Then we describe the user interface of `boxenv.sty` in Section 3. That section describes how the style file can be integrated in your LaTeX documents and how it can be customized. Section 3 concludes with an example which demonstrates the use of `boxenv.sty` within your documents. In Section 4 we describe the system interface, which consists of separate LaTeX environments for each existing BOX operator. We give examples of the use of each of these environments. Conclusions and directions for future work are described in Section 5. The implementation of the user and system interface is described in Appendix A. This part is probably of interest only for people intending to adapt, modify, or extend the package.

For a detailed description of the generic pretty-print system as a whole, and the BOX mark-up language we refer to [7].

2 The BOX Language

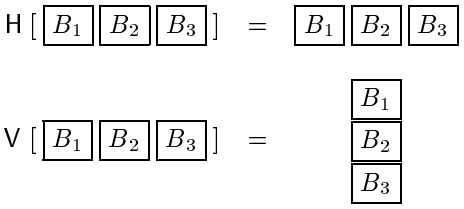
BOX is a language independent mark-up language designed to describe the intended layout of text. BOX was introduced in [9]. The BOX language supported by `box2latex` and `boxenv.sty` is based on a more recent version described in [7]. This section contains a brief overview of the BOX language as used by `box2latex` and `boxenv.sty`, and how it differs from the language described in [7]. For a comprehensive description of the language we refer to [7].

Boxes. The smallest boxes are strings. More complex boxes can be constructed by *composing* boxes using *box-operators*. BOX supports two types of operators: *positional* operators and *non-positional* operators.

operator	*description*
F	Operator to specify fonts and font attributes.
KW	Dynamic font operator to format keywords.
VAR	Dynamic font operator to format variables.
NUM	Dynamic font operator to format numbers.
MATH	Dynamic font operator to format mathematical symbols.
LBL	Operator used to define a label for a box.
REF	Operator to refer to a labeled box.
C	Operator to represent lines of comments.
L	Operator to construct a sequence of characters of arbitrary length.

Table 2: Available non-positional BOX operators.

Positional operators. Positional operators specify the relative positioning of boxes. Examples of positional BOX operators are the H and V operators, which format their sub-boxes horizontally and vertically, respectively:



The exact formatting of each BOX operator can be controlled using BOX options. For example, to control the horizontal layout between boxes, the H operator supports the hs space option:

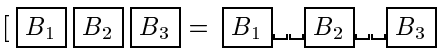
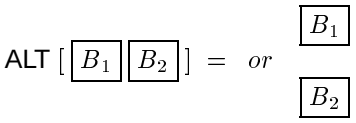


Table 2 summarizes all available positional BOX operators including their supported space options.

The BOX language that is supported by `box2latex` and `boxenv.sty` has evolved slightly since [7] because the HOV operator (which, depending on the available width, either formats all of its sub-boxes horizontally or formats them all vertically) is no longer available. Instead, the more general ALT box has been introduced. This operator formats its first sub-box when it does not exceed the right margin or otherwise, it formats its second sub-box:



Non-positional operators. The BOX language supports four types of non-positional operators: font operators to control the textual appearance of BOX expressions, cross reference operators to create links between boxes, a comment operator for formatting comments, and a line operator to construct a sequence of characters of arbitrary length. All available non-positional BOX operators are listed in Table 2.

Option	Accepted Values	Description
refstyle	*none*	No cross referencing is used.
	normal	Cross referencing is performed using the LaTeX commands `\label` and `\ref`.
	hyperref	The commands `\hypertarget` and `\hyperlink` from the package `hyperref` [5] are used for cross referencing. Select this option when your document is processed by `pdflatex` [6] in order to obtain hyperlinks in the generated PDF output.
visiblespace	*true*	Use the symbol '␣' to display spaces in quotes strings.
	false	Do not use the special symbol '␣' to display spaces in quoted strings.

Table 3: Supported options and accepted values. Options are selected by passing `option=value` to boxenv. The cross reference mechanism can also be selected using the macro `\refstyle` (see Section 3.1). The use of '␣' can be enabled/disabled using the macro `\visiblespace` (see Section 3.2).

3 User Interface

The tool `box2latex` generates a LaTeX file that represents the formatting specified in the BOX expression that was passed to `box2latex`. To use such LaTeX files in your document, you should include the `boxenv.sty` style file in your document. This file defines the macros and environments corresponding to the BOX operators.

To include the style file in your document add the following to the preamble of your document:

```
\usepackage[options]{boxenv}
```

The style file supports a number of options to customize the final output produced by LaTeX. These options are optional and can be omitted. The options together with accepted values are listed in Table 3 and are described in more detail in Section 3.1 and 3.2.

The tool `box2latex` generates a separate LaTeX file for each BOX term. These files can be included in your document using the `\input` command. For example, a generated file `f.tex` can be included using the following LaTeX command:

```
\input{f.tex}
```

After processing the file by LaTeX, a DVI file is generated which respects the formatting defined in the BOX expression(s).

font family		font series		font shape	
fm=sf	\textsf	se=md	\textmd	sh=up	\textup
rm	\textrm	bf	\textbf	it	\textit
tt	\texttt			sc	\textsc
				sl	\textsl
				em	\textem

Table 4: This table shows the BOX font options to specify font family, font series, and font shape. The table shows the accepted values, and the relation to corresponding LaTeX commands.

3.1 Cross Referencing

The `boxenv.sty` style file implements cross referencing according to the BOX operators LBL and REF (see Section 4 for a description of the relation between these BOX operators and the corresponding LaTeX code). The style file supports several cross referencing mechanisms which can be selected using the `refstyle` option. The mechanisms that are currently supported are listed in Table 3. A particular type of cross referencing can be selected by passing the option `refstyle=<reference style>`

`refstyle` to `boxenv.sty`, or by calling the macro `\refstyle{<reference style>}`. For example, to specify that `hyperlink` and `hypertarget` should be used, the option `hyperref` should be passed to `boxenv.sty` as follows:

```
\usepackage[refstyle=hyperref]{boxenv}
```

At any time a different mechanism can be selected using the `\refstyle` macro. For example, to overrule the cross reference mechanism that was selected globally one could issue the command:

```
\refstyle{normal}
```

By default, the cross reference mechanism `normal` is selected.

3.2 Visible Spaces

`visiblespace` Visualization of spaces can be controlled using the option `visiblespace` and the macro `\visiblespace`. When set to `true`, spaces occurring in a BOX string are displayed as '␣', otherwise an ordinary (invisible) space character is used. To use the visible space character, pass the option `visiblespace=true` to the style file:

```
\usepackage[visiblespace=true]{boxenv}
```

Use the macro `\visiblespace`, to overrule the globally selected space symbol. To disable the use of the visible space character, use `\visiblespace{false}`. By default, the use of '␣' is disabled.

Font operator	Corresponding LaTeX macro
KW	`\def\KWf#1{\textbf{#1}}`
VAR	`\def\VARf#1{\textit{#1}}`
NUM	`\def\NUMf#1{\textrm{#1}}`
MATH	`\def\MATHf#1{\ensuremath{#1}}`

Table 5: Dynamic font operators supported by the BOX language and their corresponding (default) LaTeX definitions. The mappings from dynamic font operators to LaTeX fonts can be redefined in the special configuration file '`box-fonts.def`' which is read by `boxenv.sty`.

3.3 Fonts

Fonts can be controlled by several BOX operators. According to [7], these font operators define fonts either statically or dynamically. Static fonts, specified using the BOX F operator, are translated by `box2latex` directly to LaTeX commands. The supported font options are based on the LaTeX commands for specifying font attributes. Font colors can be specified in BOX by name (using the cl font option) and are translated to the LaTeX `\color` command (which requires the use of the `color.sty` style file in your document). Font sizes can be specified symbolically using the sz font option. BOX uses the same symbolic name mechanism as LaTeX to specify font sizes (e.g., `tiny`, `large`, etc.). The remaining font options and their relation to LaTeX commands are displayed in Table 4.

Dynamic font operators are interpreted when processed by LaTeX. The `boxenv.sty` style file contains default mappings from these operators to LaTeX fonts. For example, text defined within the VAR font operator (`VAR[some-text]`) translates by default to italics (`\textit{some-text}`).

The style package allows these mappings to be customized manually by redefining the mappings in a configuration file (the file '`box-fonts.def`') somewhere in your `TEXINPUTS` search path. When this file exists it is loaded and its definitions override the default definitions.

For example, to change the default mapping of the VAR font operator, we can add a new macro definition to the file `box-fonts.def`:

```
\def\VARf#1{{\color{red}#1}}
```

This definition of `VARf` results in a red coloring of text specified within a VAR font operator. In Table 5 all dynamic font operators of the BOX language and the corresponding (default) LaTeX macros are listed.

3.4 Example

In this section we demonstrate the use of the `boxenv.sty` style file and the integration of files generated by `box2latex` within your document according to the LaTeX file depicted in Figure 1.

```
\documentclass{article}
\usepackage[refstyle=none,visiblespace=false]{boxenv}

\begin{document}
   \section{File1}
       \input{File1.tex}
   \section{File2}
       {
          \refstyle{hyperref}
          \visiblespace{true}
          \input{File2.tex}
       }
   \section{File3}
       \input{File3.tex}
\end{document}
```

Figure 1: A sample LaTeX document to demonstrate the use of `boxenv.sty`.

The LaTeX document of Figure 1 uses the `boxenv.sty` style file. The options `refstyle` and `visiblespace` are used to disable cross referencing and the use of '␣' as space character. The file contains three sections each of which imports a file (generated by `box2latex`). The first file is typeset according to the options specified in the documents preamble (i.e., no cross referencing and no '␣' as space character). Before the second file is processed by LaTeX, cross referencing and visiblespace are enabled. Thus, the symbol '␣' will be used as space character and the macros `hypertarget` and `hyperlink` will be used for cross referencing during formatting of the second file. Because both macros are called within a LaTeX environment, their values are restored when this environment is closed. Hence, the third file is again formatted without cross referencing and without using '␣'.

4 System Interface

The `boxenv.sty` style file defines a number of high-level macros/environments, and low-level macros/environments that form their implementation. This section describes the relation between BOX operators and their corresponding high-level LaTeX environments and macros. Implementation details and a description of the low-level interface are postponed to Appendix A.

The high-level environments and macros correspond directly to BOX operators and form the system interface of `boxenv.sty`. For each BOX operator a corresponding environment or macro is defined. One exception is formed by the BOX comment operator for which no corresponding macro or environment is defined (see the description of the `latextext` environment below).

boxenv All environments and macros described in the remaining of this section are defined within the boxenv environment. This environment initializes the BOX related environments as well as several formatting parameters. Furthermore, it is responsible for saving several TEX/LATEX parameters when this environment is entered and restoring them upon exit.

The boxenv environment has one optional parameter: the desired maximum line width. BOX operators that take the line width into account (i.e., the HV and ALT operators) will break a line into lines of smaller width when the desired line width is exceeded. Within the latextext environment (see below), the width parameter is used to format paragraphs of text of this width.

When the width parameter is not specified \linewidth is used as default value.

Example:

```
\documentclass{article}
\usepackage{boxenv}

\begin{document}

   \begin{boxenv}[.5\linewidth]
      ...
   \end{boxenv}

\end{document}
```

HBOX The environment HBOX corresponds to the BOX operator H. Text within the HBOX environment is formatted horizontally without line breaking. In the current implementation, line breaking is not switched off by the HBOX environment. Instead, line breaking should be prevented by using non-breakable spaces. This may change in a future release of boxenv.sty. This environment requires one argument: the inter-word spacing factor (corresponding to the hs space option).

Example:

```
\begin{HBOX}{2}
   An~example~of~the~use~of~the~HBOX~environment.~%
   Note~the~use~of~non-breakable~spaces~to~prevent~%
   line~breaking.
\end{HBOX}
```

VBOX The VBOX environment corresponds to the V operator. Paragraphs of text within a VBOX environment are placed vertically. When entering a VBOX environment, the left margin is set to the current horizontal position. As a result, subsequent paragraphs are placed exactly below \begin{VBOX}. Observe that text is not placed vertically automatically. To force a line to be placed below another, either a new paragraph should be started or an explicit line break should be inserted.

The VBOX environment requires two arguments. The first argument specifies the inter-line space factor (according to the vs space option). It is used to specify the vertical distance between subsequent lines. The second argument specifies the indentation

of lines (according to the is space option). Note that the is space option defines layout between sub boxes. As a consequence, indentation within the VBOX environment starts at the second paragraph.

Example:

```
\begin{VBOX}{0}{3}
   Within this environment normal inter-line spacing is\\
      used. Subsequent lines are indented 3 units \\
      (currently this corresponds to 3 ex).
\end{VBOX}
```

HVBOX The HVBOX environment corresponds to the HV BOX operator. Within this environment line breaking is used whenever possible to construct lines that do not exceed the right margin of the surrounding boxenv environment. Since some text within the HVBOX environment might not be breakable (for instance text within the HBOX environment), the breaking mechanism does not guarantee to succeed. This may result in lines that still exceed the right margin.

Because the HVBOX environment is a combination of horizontal and vertical placement of text (i.e., the HBOX and VBOX environments), it requires three arguments. The first argument specifies the inter-word space factor (corresponding to the BOX hs space option). The second argument specifies the inter-line spacing factor (corresponding to the vs space option). The last argument specifies the indentation factor (corresponding to the is space option).

Example:

```
\begin{HVBOX}{1}{0}{0}
   Text within this environment has normal inter-word and
   inter-line spacing (hs=1 and vs=0). Subsequent lines
   are not indented (is=0).
\end{HVBOX}
```

ALTBOX The ALTBOX environment corresponds to the BOX ALT operator. The ALTBOX environment requires two arguments. When the width of the text corresponding to the first argument does not exceed the desired width of the surrounding boxenv environment, that text is used. Otherwise, the text corresponding to the second argument is used.

Example:

```
\begin{ALTBOX}{%
   \begin{HBOX}{1}
      This~text~is~preferred~when~it~fits~on~a~single~line%
   \end{HBOX}%
   }{%
   \begin{VBOX}{0}{0}
      Otherwise, this text is used
   \end{VBOX}%
   }
\end{ALTBOX}
```

Alignment	Generated template
left	#\hfill
right	\hfill#
center	\hfill#\hfill

Table 6: Templates generated by box2latex for left, right, and centered alignment.

ABOX The ABOX environment corresponds to the BOX A operator. This environment is implemented using the alignment mechanism of TEX (see [4] and [3] for a description of TEXs alignment). As a consequence, text within this environment should accomplish the syntax of alignments in TEX (i.e., columns should be separated by '&', rows should be separated by '$\backslash cr$'). This environment requires a single argument that specifies the template of the alignment. The templates that are generated by the tool box2latex for left, right and centered alignment, corresponding to the l, r, and c alignment options are depicted in Table 6.

Example:

```
\begin{ABOX}{#&~\hfill#\cr}
   first  & row\cr
   second & row\cr
\end{ABOX}
```

LBOX This environment corresponds to the BOX L line operator. It draws a line of width approximately equal to the width of the text t within the LBOX environment. The LBOX environment requires one argument that specifies the string s that is used to construct the line. When s equals '=' the LATEX command \hrule is used to construct the line. In this case the width is equal to the width of t. In all other cases, a line is constructed by taking n copies of s such that $(n-1) \times \mid s \mid < \mid t \mid \leq n \times \mid s \mid$.

Example:

```
\begin{LBOX}{=}
   This would draw a line using \hrule of width exactly
   equal to this text.
\end{LBOX}
```

latextext Text within this environment is formatted as ordinary text. That is, all TEX/LATEX parameters that are modified within the boxenv environment are restored. The only exceptions are the \hsize and the \linewidth parameters. The latextext environment thus respects the width of the surrounding boxenv environment. The latextext environment has no corresponding BOX operator. The environment is used by the box2latex tool to format text within the BOX C operator. Strings occurring inside a C operator are placed in a latextext environment by the tool after the initial comment characters ('%%') are removed.

Example:

```
\begin{latextext}
   Except for the right margin, all parameters modified
   within the boxenv environment are restored in the
   latextext environment.
\end{latextext}
```

boxlabel The macro `\boxlabel` corresponds to the BOX operator LBL. The macro is used to define a new label. The exact LaTeX code to which the macro expands depends on the cross reference style that is selected (see Section 3). The macro has two arguments. The first argument specifies the name of the label. The second argument specifies the text that is to be labeled.

Example:

```
\boxlabel{mylabel}{Text that should be labeled}
```

boxref The macro `\boxref` corresponds to the BOX operator REF and is used to reference to some label. Undefined references are discarded. The way that references are displayed depends on the cross reference mechanism that is selected (see Section 3). When '`refstyle=none`' is specified, referencing is disabled. The default (i.e., '`refstyle=normal`') is to display defined references using the LaTeX command `\ref`. How references are displayed when the style `hyperref` is selected ('`refstyle=hyperref`'), depends on the options passed to the style package `hyperref` [5] and whether or not the LaTeX file is processed by `pdflatex` [6].

The macro requires two arguments. The first argument specifies the name of the label to refer to. The second argument specifies the text to display as label.

Example:

```
\boxref{the label to refer to}{Text displayed as label}
```

4.1 Example

In this section the use of some of the environments defined in `boxenv.sty` is demonstrated by means of a number of small examples. This section is not intended to provide a complete demonstration of the `boxenv.sty` style file, but rather to give some intuition about how to use the environments and what the resulting output looks like.

HBOX. Horizontal text can be formatted using the HBOX environment. The LaTeX code below forms a small example of the use of this environment:

```
\begin{HBOX}{5}%
   a~line~of~horizontal~text%
\end{HBOX}
```

This example results in a single line of text. Individual words are separated by a rather large amount of horizontal space because an inter-word value of 5 (which corresponds to 5ex) was passed as argument to the environment. When processed by LaTeX the following result is obtained:

a line of horizontal text

VBOX. Text can be formatted vertically using the VBOX environment. Each paragraph within this environment is placed below the preceding paragraph and is optionally indented to the right. The left margin within a VBOX environment equals the horizontal position where the VBOX starts:

```
\begin{HBOX}{1}%
   some~prefix.~%
   \begin{VBOX}{0}{3}%
      \begin{HBOX}{1}a~line~of~horizontal~text\end{HBOX}%

      \begin{HBOX}{1}a~second~line~of~text\end{HBOX}%

      \begin{HBOX}{1}a~third~line~of~text\end{HBOX}%
   \end{VBOX}%
\end{HBOX}
```

This example shows the composition of a VBOX environment within a HBOX environment. The VBOX environment is preceded by horizontal text and, as a consequence, the left margin within the VBOX environment is set to the position after this text. The VBOX environment contains three lines of text, each line is formatted horizontally (because the corresponding text is placed within an HBOX environment). There is no extra vertical space inserted between subsequent lines. There is horizontal layout inserted between lines due to the non-zero value passed as argument to the environment. Consequently, the second and third lines are indented to the right:

some prefix. a line of horizontal text
a second line of text
a third line of text

LBOX. Width calculation of text is performed by the LBOX environment. In the example below a vertical composition is constructed consisting of a line of text formatted horizontally, and a horizontal bar of equal width:

```
\begin{VBOX}{0}{0}
   \begin{HBOX}{1}a~line~of~horizontal~text\end{HBOX}%

   \begin{LBOX}{=}%
      \begin{HBOX}{1}a~line~of~horizontal~text\end{HBOX}%
   \end{LBOX}%
\end{VBOX}%
```

LaTeX formats and calculates the width of the text within the LBOX environment but without outputting the formatted text. Instead, it constructs a bar of equal width. Because the HBOX and LBOX are placed within a vertical environment, both the text and the horizontal bar are positioned below each other:

a line of horizontal text

The LBOX environment constructs a sequence of symbols as passed as argument to the environment. When the symbol '=' is passed (as is the case in the example), a horizontal bar is constructed using the `\hrule` command.

5 Concluding Remarks

The tool `box2latex` translates BOX expressions to LaTeX code by replacing BOX operators to corresponding LaTeX environments. The `boxenv.sty` style file contains the implementation of these environments and macros. This style file is required in order to be able to process the generated files by LaTeX.

This article gives a brief overview of the BOX mark-up language and it describes the use and implementation of the `boxenv.sty` style file.

Current Status and Limitations. The combination of `box2latex` and the style file `boxenv.sty` currently implements most BOX operators. The following features are not supported at this moment:

- The F font operator does not support font name selection using the fn font option.
- The hs and is space options cannot be specified as BOX expressions. This means that a BOX expression like $\mathsf{H}_{\mathsf{hs}=\mathsf{H}\ [\ldots]}\ [\ldots]$ cannot be translated to LaTeX.

In addition to these missing features there are also some semantical issues that have not yet been solved. These include the formatting of non-horizontal BOX operators within an alignment operator. At this moment we therefore do not support the V, ALT, and HV operators within an alignment.

Another problem is formed by the page breaking algorithm. This algorithm is based on the heuristic that a page break may occur only within a vertical context when the vs space option is greater then zero. That is, we allow a page break to occur whenever empty lines are inserted between subsequent boxes in a vertical context. Unfortunately, this algorithm does not work correctly at the moment and needs improvement.

Future Work. Although the translation of BOX to LaTeX is almost finished there are still some remaining issues. These include the implementation of the missing features, solving the semantical problems, and improving the page breaking mechanism. Furthermore, we want to investigate the translation from BOX to HTML using the translator `latex2html`. Although generating HTML in this way already works, the result is not satisfactory because the `latex2html` translator generates images from the formatted BOX expressions (instead of using HTML tags).

Acknowledgments. We would like to thank Mark van den Brand (CWI), Paul Klint (CWI), and Joost Visser (CWI) for reading earlier drafts of this paper.

References

[1] D. Carlisle. The keyvalid package. available at `ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex/`, 1998.

[2] Victor E. Unusual paragraph shaped. *TUGboat*, 11(1):51–53, 1990.

[3] Victor E. *TEX by Topic a TEXnician's Reference*. Addison-Wesley, 1991.

[4] D. E. Knuth. *The TEXbook*, volume A of *Computers & Typesetting*. Addison-Wesley, 1984. (Ninth printing, revised, October 1989).

[5] S. Rahtz. Hyperref. available at `http://www.tex.ac.uk/tex-archive/macros/latex2e/contrib/supported/hyper%ref/`, 1998.

[6] T. H. Thanh. A TEX variant which can produce acrobat pdf instead of dvi. available at `ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex/`, 1998.

[7] M. G. J. van den Brand and M. de Jonge. Pretty Printing within the ASF+SDF Meta-Environment: a Generic Approach. Technical Report SEN-R9904, CWI, 1999.

[8] M. G. J. van den Brand, P. Klint, P. Olivier, and E. Visser. AsFix, 1997. A structured data format for representation of parse trees with an extensive library with generic, language independent functionality. In particular a format for ASF+SDF specifications. (In preparation).

[9] M. G. J. van den Brand and E. Visser. Generation of formatters for context-free languages. *ACM Transactions on Software Engineering and Methodology*, 5(1):1–41, 1996.

A Implementation

This section describes the implementation of the boxenv style file. It describes the implementation of the user and system interface introduced in Section 3 and Section 4, as well as the environments and macros that are used by the interfaces.

The translation from BOX to LaTeX requires the ability to type-set boxes horizontally and vertically (with optional left indentation), and the ability to calculate the width of boxes.

Horizontal placement and indentation is implemented by a new environment (box@hangpar) that supports paragraphs with a hanging indentation. Entering a box@hangpar environment marks the left margin of subsequent paragraphs. Closing a box@hangpar environment restores the left margin. By nesting box@hangpar environments, one can gradually increase and decrease the indentation of lines and paragraphs.

Calculation of the width of paragraphs is performed by the box@width environment and is required for the ALT and L operators. This calculation is rather complicated and explained in detail in Section A.9. Since almost all BOX operators can be expressed in terms of paragraphs with hanging indentation and width calculations, their implementation is straightforward by using the box@hangpar and box@width environments.

All macros and variables that do not belong to the user interface of boxenv.sty (see Section 3) are prefixed with box@ to make them inaccessible outside the style file. Furthermore, to enforce that BOX environments are used from within the boxenv environment only, all environments test whether or not they are embedded within the boxenv environment.

A.1 Version

Code to specify author, the version of boxenv, and last modification date.

```
\def\@fileversion{1.0}
\def\@filedate{1999/04/27}
\def\@author{Merijn de Jonge (mdejonge@cwi.nl)}
```

Identification of package file:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{boxenv}

\typeout{Package: `boxenv'
   \@fileversion\space <\@filedate> (\@author)}
```

A.2 Processing Options Passed to boxenv

The refstyle and visiblespace options can be passed as key/value pairs to boxenv. For this value passing mechanism we used the keyvald package of [1].

```
\RequirePackage{keyval}
```

We define a new set (‘Boxenv’) of key/value pairs that contains the keys ‘refstyle’ and ‘visiblespace’, and we use the macros \refstyle and \visiblespace to process

the values corresponding to these options (see Table 3 for the list of values currently accepted for both options).

```
\define@key{Boxenv}{refstyle}{%
\refstyle{#1}%
}
\define@key{Boxenv}{visiblespace}{%
\visiblespace{#1}%
}
```

refstyle The macro `\restyle` selects a cross reference mechanism according to its argument (see Table 3 for the list of mechanisms currently available). For each mechanism m two macros are defined `\box@`m`label` and `\box@`m`ref`. When processing the selected style we link the macros corresponding to the selected style to the macros `\boxlabel` and `\boxref`.

```
\def\refstyle#1{%
\def\@tmpa{#1}%
\def\@normal{normal}%
\def\@none{none}%
\def\@hyperref{hyperref}%
```

The option '`refstyle=normal`' has been selected. We link `\box@normallabel` and `\box@normalref` to `\boxlabel` and `\boxref`, respectively.

```
\ifx\@tmpa\@normal%
   \def\boxlabel{\box@normallabel}%
   \def\boxref{\box@normalref}%
```

The option '`refstyle=none`' has been selected. We link `\box@nonelabel` and `\box@noneref` to `\boxlabel` and `\boxref`, respectively.

```
   \else%
   \ifx\@tmpa\@none%
   \def\boxlabel{\box@nonelabel}%
   \def\boxref{\box@noneref}%
```

The option '`refstyle=hyperref`' has been selected. We link the macros `\box@hyperlabel` and `\box@hyperref` to `\boxlabel` and `\boxref`, respectively.

```
      \else%
      \ifx\@tmpa\@hyperref%
         \def\boxlabel{\box@hyperreflabel}%
         \def\boxref{\box@hyperrefref}%
```

Display an error message when an invalid value has been specified.

```
      \else%
         \errmessage{Invalid refstyle option passed.}%
      \fi%
   \fi%
\fi%
}
```

boxspace The macro \boxspace is used to display significant spaces. Using the macro \visiblespace, this macro can be redefined to use either an ordinary (invisible) space (the symbol '~'), or the symbol '␣'. By default the symbol '~' is used.

```
\def\boxspace{~}%
```

visiblespace The macro \visiblespace is used to select the symbol that is used by LaTeX to display a space character (see Section 3.2 for a description of this macro).

When the value true is passed to \visiblespace, we redefine the macro \boxspace in order to display spaces as '␣'. Otherwise, we use an ordinary (invisible) space (using the LaTeX character '~').

```
\def\visiblespace#1{%
\def\@tmpa{#1}%
\def\@true{true}%
\def\@false{false}%
\ifx\@tmpa\@true%
\def\boxspace{\hbox{\tt\char'\ }}%
\else%
\def\boxspace{~}%
\fi%
}
```

ProcessOptionsWithKV In order to be able to specify a key/value pair to '\usepackage', we define the macro \ProcessOptionsWithKV. This macro is copied from the hyperref package [5].

```
\def\ProcessOptionsWithKV#1{%
  \let\@tempc\relax
  \let\@tempa\@empty
  \@for\CurrentOption:=\@classoptionslist\do{%
     \@ifundefined{KV@#1@\CurrentOption}%
       {}%
       {\edef\@tempa{\@tempa,\CurrentOption,}}}%
     \edef\@tempa{%

   \noexpand\setkeys{#1}%
      {\@tempa\@ptionlist{\@currname.\@currext}}}%
       \@tempa
     \AtEndOfPackage{\let\@unprocessedoptions\relax}%
}
```

Finally, we process the options in the set of key/value pairs 'Boxenv'.

```
\ProcessOptionsWithKV{Boxenv}
```

A.3 Variable Declarations

The variable \box@leftmargin defines the left margin in a box@hangpar environment. It corresponds to the horizontal offset from the beginning of the box@hangpar environment. \box@indentation corresponds to the is space option passed to the V and HV operators. Both variables are used after starting a new paragraph to set the value of \leftskip and \hangindent, respectively.

```
\newdimen\box@leftmargin
\newdimen\box@indentation
```

The variables below are used to store the results of `\box@currentxpos` and `\box@width`, and to store temporary results during width calculation.

```
\newdimen\box@xpos
\newdimen\box@thewidth
\newdimen\box@widthincalculation
```

Inside the `\box@width` environment line widths are computed in order to calculate the current x position. The variable `\box@lastlinewidth` contains the width of the last line of the most recent paragraph

```
\newdimen\box@lastlinewidth
```

The variables below are used to save LaTeX settings when entering the `boxenv` environment. They are restored when normal text has to be formatted within the `latextext` environment.

```
\newdimen\box@linewidth
\newdimen\box@rightskip
\newdimen\box@parindent
\newcount\box@hyphenpenalty
```

The boolean `\ifbox@inboxenv` is used to test whether or not we are inside a `box@hangpar` environment

```
\newif\ifbox@inboxenv
```

The three variables defined below are required in the macros `\box@traverselines` and `\box@thelinewidth`, and in the environment `\box@width` for calculating the length of paragraphs.

```
\newbox\box@investigation
\newbox\box@tester
\newbox\box@widthcalculation
```

Finally, a global variable is required that is used by `\box@hspaceskip` and `\box@vspaceskip` to store their return value.

```
\newskip\box@tmpskip
```

A.4 Space Options

According to the BOX space options we define two macros that calculate the horizontal and vertical spaceskip between words and lines, respectively.

```
\newcommand{\box@vspaceskip}[1]{\box@tmpskip=#1ex}%
```

```
\newcommand{\box@hspaceskip}[1]{%
\dimen0=#1em%
\dimen0=.3333\dimen0%
\box@tmpskip=\dimen0 plus \fontdimen3\font%
}%
```

A.5 Fonts

KWf
VARf
NUMf
MATHf
COMMf

As described in Section 3, the `boxenv.sty` file contains mappings from BOX font operators to LaTeX fonts. The following mappings are defined:

```
\def\KWf#1{\textbf{#1}}%
\def\VARf#1{\textit{#1}}%
\def\NUMf#1{\textrm{#1}}%
\def\MATHf#1{\ensuremath{#1}}%
\def\COMMf#1{\textrm{#1}}%
```

In order to allow a user to redefine these mappings to meet his particular needs without modifying `boxenv.sty`, he may define mappings in a configuration file '`box-fonts.def`'. The existence of this file is optional. When missing the default macros are used.

```
\IfFileExists{box-fonts.def}{%
\input{box-fonts.def}%
\typeout{Using box-font definitions
   from ''box-fonts.def''}%
}{%
\typeout{Using default box-font definitions.}%
}%
```

A.6 Implementation User Interface

boxenv

This environment initializes several parameters of the `boxenv` style file. Therefore, each BOX environment has to be embedded within `boxenv.sty`. The environment supports one optional argument (see Section 3).

```
\newenvironment{boxenv}[1][\linewidth]{%
```

First, we save several TeX/LaTeX parameters. They are restored when entering the `latextext` environment in order to be able to type-set normal text.

```
\box@linewidth=\linewidth%
\box@rightskip=\rightskip%
\box@hyphenpenalty=\hyphenpenalty%
\box@parindent=\parindent%
```

Next, the text width (parameters `\hsize` and `\linewidth`) is set according to the optional argument to `boxenv`:

```
\linewidth=#1%
\hsize=#1%
```

Initialization of variables is the next step. Observe that we configure a right skip with a rubber length to allow a ragged right margin.

```
\box@leftmargin=\z@%
\box@indentation=\z@%
\rightskip=0pt plus 1 fill%
\parindent=0pt%
```

After initialization, we leave vertical mode and we set the flag `\box@inboxenv` to `true`. This flag is inspected by other BOX-environments to verify that they are embedded within the `boxenv` environment.

```
\leavevmode%
\box@inboxenvtrue%
```

We end the current paragraph explicitly when leaving the environment.

```
}{\par}%
```

HBOX In Section 3 we described that the current implementation of `boxenv.sty` does not prevent inter-word line breaking and that it should be prevented by the `box2latex` tool (by using non-breakable spaces). The current implementation only prevents breaking of lines and pages at hyphens.

This implementation of the HBOX environment therefore consists of the configuration of penalty parameters (`\penalty`, `\linepenalty`, and `\hyphenpenalty`) and the configuration of inter-word spacing according to the parameter passed to the environment.

```
\newenvironment{HBOX}[1]{%
\box@testforboxenv%
\penalty10000%
\linepenalty10000%
\hyphenpenalty=10000%
\box@hspaceskip{#1}\spaceskip=\the\box@tmpskip%
}{}%
```

VBOX See Section 3 for a description of the environment and its parameters. The VBOX environment is implemented as a `box@hangpar` environment. Before entering the `box@hangpar` environment we verify that the VBOX environment is enclosed by a `boxenv` environment.

```
\newenvironment{VBOX}[2]{%
\box@testforboxenv%
\begin{box@hangpar}{#1}{#2}%
}{%
\end{box@hangpar}%
}%
```

HVBOX See Section 3 for a description of the environment, its parameters, and its use. The implementation of the HV environment is similar to the implementation of the V environment, by using the `box@hangpar` environment. But, unlike the V environment, the HV environment also configures the inter-word spacing for the horizontal formatting of text.

```
\newenvironment{HVBOX}[3]{%
\box@testforboxenv%
\box@hspaceskip{#1}\spaceskip=\the\box@tmpskip%
\begin{box@hangpar}{#2}{#3}%
}{%
\end{box@hangpar}%
}%
```

ALTBOX The ALTBOX environment is implemented using the `\box@width` environment. First the width w of the text passed as argument `#1` is calculated. Next, w is compared to the maximal allowed width max (i.e., `\linewidth` $-$ `\box@xpos`). When $w \leq max$, the text passed as argument `#1` is used, otherwise `#2` is used.

```
\newenvironment{ALTBOX}[2]{%
\box@testforboxenv%
\ifhmode\null\fi%
\box@currentxpos%
\dimen0=\linewidth%
\advance\dimen0-\box@xpos%
\edef\remainingwidth{\the\dimen0}%
\edef\saved@xpos{\the\box@xpos}%
\begin{box@width}%
#1%
\end{box@width}%
\hskip\saved@xpos%
\hskip-\box@xpos%
\ifdim\box@thewidth>\remainingwidth%
#2%
\else%
#1%
\fi%
}{}%
```

ABOX The ABOX environment is implemented as a TEX alignment (see Section 3 for a description of ABOX). The `box2latex` tool should construct a suitable string defining the number and alignments of columns and the spacing between columns (according to the hs space option). Furthermore, this tool is responsible for inter-line spacing (the vs space option).

```
\newenvironment{ABOX}[1]{%
\box@testforboxenv%
\tabcolsep=0pt
\box@hspaceskip{1}\spaceskip=\the\box@tmpskip%
\begin{box@hangpar}{0}{0}%
\halign\bgroup%
\hskip\box@indentation\hskip\box@leftmargin#1%
}{%
\egroup%
\end{box@hangpar}%
}%
```

LBOX The LBOX environment draws a line or a sequence of characters according to the argument passed to the environment (see Section 3). The width of the line (or sequence of characters) equals the width of the text within the environment. The environment is implemented using the `\box@width` environment which calculates the width (`\box@thewidth`) of the text in the environment and the macro `\box@wcopies` which draws a line or sequence of characters.

```
\newenvironment{LBOX}[1]{%
```

```
\box@testforboxenv%
\def\char{#1}%
\begin{box@width}%
}{%
\end{box@width}%
\box@wcopies{\box@thewidth}{\char}%
}%
```

latextext The `latextext` environment is used to type-set ordinary text within a `boxenv` environment. It is normally used to type-set text specified in C boxes after the initial comment characters '%%' have been removed. Parameters that were modified by BOX macros are restored when entering this environment. These parameters include the left margin, penalties, `\spaceskip`, and `\hsize`. The text is formatted as a separate paragraph (i.e., `latextext` starts and ends with an explicit `\par` command).

```
\newenvironment{latextext}{%
\box@testforboxenv%
\let\par\endgraf%
\leftskip=0pt%
\rightskip=0pt%
\hsize=\box@linewidth%
\advance\hsize-\box@leftmargin%
\hangindent=0pt%
\hyphenpenalty=50%
\par%
\spaceskip=0pt%
\parindent=\box@parindent%
\ifhmode%
\vbox%
\fi%
\bgroup%
}{%
\egroup%
```

We insert the command `\hidewidth` for the case that the text occurs within an alignment. By doing so, the width of the text within the `latextext` environment does not affect the width of entries within alignments (it will stick out to the right). In this way, comments can be used within the A BOX operator without any problem. Observe that `\hidewidth` is only inserted in horizontal mode.

```
\ifhmode%
\hidewidth%
\fi%
\par%
}%
```

A.7 Cross Referencing

This section describes the implementation of the cross referencing mechanism. It contains a description of the three different mechanisms that are currently implemented (see Table 3).

boxlabel
boxref

The macros \boxlabel and \boxref are linked to different macros depending on the 'refstyle' option passed to boxenv. By default they are linked to \box@normallabel and \box@normalref, respectively.

```
\@ifundefined{boxlabel}{\def\boxlabel{\box@normallabel}}{}
\@ifundefined{boxref}{\def\boxref{\box@normalref}{}}
```

box@nonelabel
box@noneref

The macros \box@nonelabel and \box@noneref are used when the option 'refstyle=none' is passed to boxenv and disable the cross reference mechanism.

```
\long\def\box@nonelabel#1#2{#2}
\long\def\box@noneref#1#2{#2}
```

box@normallabel
box@normalref

Both macros are used when 'refstyle=normal' is passed to boxenv. These macros implement a cross reference mechanism using the LaTeX commands \label and \ref.

```
\long\def\box@normallabel#1#2{\label{#1}#2}
```

A reference is only displayed when the corresponding label is defined. When no label has been defined the reference is discarded.

```
\long\def\box@normalref#1#2{%
#2%
\@ifundefined{r@#1}{}{%
$^{\mbox{\tiny\ref{#1}}}$}%
}
```

Note that the labels are displayed in super-script using \tiny font.

box@hyperreflabel
box@hyperrefref

These macros are used when 'refstyle=hyperref' has been specified. When this option is specified, labeling and cross referencing is implemented using the macros \hypertarget and \hyperlink, respectively. This style of cross referencing is useful especially when the LaTeX document is processed by pdflatex [6] to obtain an interactive document. Cross references are then implemented as hyper-links. We refer to [5] for a description of the macros \hypertarget and \hyperref. How the labels and references are displayed in the final document is described in [5] as well.

```
\long\def\box@hyperreflabel#1#2{\hypertarget{#1}{}#2}
\long\def\box@hyperrefref#1#2{\hyperlink{#1}{#2}}
```

A.8 The hangpar Environment

This section describes the implementation of the box@hangpar environment and the macro \box@currentxposition that is used to implement the environment.

hangpar

The box@hangpar environment is the most important building block of the boxenv style file. It implements an environment in which subsequent paragraphs are left indented according to the horizontal position where the box@hangpar was entered.

For example:

```
a \begin{hangpar}{0}{0}b\\
    c
  \end{hangpar} d
```

is formatted as

```
a b
  c d
```

The environment starts a new paragraph but does *not* start a new line. Likewise, text that follows a box@hangpar environment is not placed on a new line.

Left indentation of subsequent lines is implemented by defining a new \par macro. This macro sets \leftskip to the x position where the environment was started. Furthermore, \hangindent is set according to the indentation factor passed as argument to the environment. The indentation factor corresponds to the is space option of the V and HV BOX operators.

```
\newenvironment{box@hangpar}[2]{%
\box@testforboxenv%
\ifhmode\null\fi%
```

Initialization of box@hangpar is performed in a number of steps. First, the current x position is determined (using \box@currentxpos, see below) and saved in box@savedxpos because it might be changed by \par. Furthermore a new paragraph is started.

```
\box@currentxpos%
\edef\box@savedxpos{\the\box@xpos}%
\parskip=0pt%
\parshape=0%
\par%
```

After issuing the \par command we reset the current x position.

```
\global\box@xpos\box@savedxpos%
```

Next, we define a macro (\newpar) that extends the behavior of the old \par macro. When the definition of \par is different from \newpar, we save the definition of \par in \oldpar. The old definition is used in \newpar to 'chain' different extensions of the \par macro. Initially \oldpar equals \endgraf. Starting a new paragraph therefore results in the chain \newpar $\rightarrow$ \endgraf. In general, whenever a chain of extensions $\epsilon_0 \ldots \epsilon_i$ exists for the \par macro, the chain is extended to obtain $\epsilon_0 \ldots \epsilon_i \epsilon_{i+1}$.

```
\def\newpar{%
\dimen0\hangindent%
\oldpar%
\box@indentation=\the\dimen0%
\box@setpenalty%
\leftskip=\box@leftmargin%
\hangindent=\box@indentation%
\hangafter=0%
```

A \par command starts a new line, hence we reset the value of \box@xpos to box@leftmargin + box@indentation.

```
\global\box@xpos\box@leftmargin%
\global\advance\box@xpos\box@indentation%
}%
\ifx\par\newpar%
\else%
\let\oldpar=\par%
\fi%
\let\par\newpar%
```

Next, the vertical skip between paragraphs is configured according to the first argument of box@hangpar. This corresponds to the vs space option of the V and HV operators. Because the vertical space option should only affect boxes within the V and HV boxes, the initial parskip is undone by an extra \vskip.

```
\box@vspaceskip{#1}\parskip=\the\box@tmpskip%
\vskip-\parskip%
```

Indentation according to the is space option (which is passed as second argument to box@hangpar) is implemented using \hangindent. Since the value of this macro is reset by LaTeX after each new paragraph start, the desired left indentation is saved in \box@indentation and used in \newpar to configure \hangindent after a paragraph start. Note that the is space option specifies extra white space *between* boxes. This white space should therefore *not* be put in front of the first line of text within the box@hangpar environment. For this reason \hangafter is set to 1 to take affect after the first line and setting \box@indentation is delayed until the next paragraph (its future value is store in \hangindent).

```
\box@hspaceskip{#2}\hangindent=\box@tmpskip%
\box@indentation=\z@%
\hangafter=1%
```

The last initialization step of the box@hangpar environment is the configuration of the left margin. It is configured by setting \leftskip to the current x position. Because \leftskip is reset by LaTeX during a paragraph start, we save the current x position in \box@xpos. Its value is used in \newpar to re-configure \leftskip.

```
\box@leftmargin=\box@xpos%
\leftskip=\box@xpos%
}
```

Ending a box@hangpar environment should restore the left margin. Furthermore, whenever the box@hangpar environment is ended in horizontal mode, we should accomplish that text following the box@hangpar environment is continued on the same line. Whenever the box@hangpar environment is ended in vertical mode, text following the environment has to be placed under the environment and no indentation is required.

To restore the left margin in horizontal mode a new paragraph should be started to prevent LaTeX from using this left margin for the current paragraph as well. \box@currentxpos is used to obtain the current x position and (as a side effect) to start a new paragraph. \box@leftindent is used to accomplish that text starting the new paragraph continues on the current line.

```
{%
\ifhmode%
\box@currentxpos%
\box@leftindent\box@xpos%
\fi%
}
```

box@currentxpos The macro \box@currentxpos determines the current x position. The current x position is the horizontal position where the macro \box@currentxpos occurs in

the text (after type setting). More precisely, the current x position equals the width of the last line of the paragraph that is ended by `\box@currentxpos`.

We have implemented `\box@currentxpos` using the `\predisplaysize` macro that gives access within a display to the width of the previous line. A display ends the current paragraph and, as a consequence, ending the current paragraph is a side effect of `\box@currentxpos`.

```
\newcommand{\box@currentxpos}{%
\begingroup%
```

Before opening a display, we disable page breaking before and after the display by setting `\predisplaypenalty` and `\postdisplaypenalty` to 10000.

```
\predisplaypenalty10000%
\postdisplaypenalty10000%
$$%
```

According to [4, page 188] `\predisplaysize` contains the width of the line preceding the display plus two ems in the current font. However, when the length of that line depends on glue being stretched or shrunken, `\predisplaysize` is set to `\maxdimen`. Finally, if there was no previous `\line`, `\predisplaysize` is set to `-\maxdimen`. According to the value of `\predisplaysize` we set `\box@xpos` to `\predisplaysize` $- 2em$, `\linewidth`, and zero, respectively.

```
\ifdim\predisplaysize=-\maxdimen%
 \global\box@xpos=\z@%
\else
 \ifdim\predisplaysize=\maxdimen%
  \global\box@xpos=\linewidth%
 \else%
  \global\box@xpos=\predisplaysize%
  \global\advance\box@xpos-2em%
 \fi%
\fi%
```

A display is assumed to take three lines [4, page 188]. We use negative display skips to prevent `\box@currentxpos` from occupying these lines.

```
\abovedisplayskip-\baselineskip%
\belowdisplayskip-\baselineskip%
\abovedisplayshortskip-\baselineskip%
\belowdisplayshortskip-\baselineskip%
```

Finally, we end the display and the group and we are done.

```
$$%
\endgroup%
}%
```

A.9 The box@width Environment

This section documents the implementation of the `\box@width` environment and several macros that are required for this implementation.

The `\box@width` environment is able to calculate the maximum width of paragraphs of text embedded in this environment. The width is stored in the global variable `\box@thewidth`.

The algorithm for the calculation of the width of text is as follows: After each paragraph end (by a `\par` command), we traverse the horizontal boxes constituting the lines of the last paragraph (using `\lastbox`). Since LaTeX keeps track of the width of these boxes, we can obtain the width of each individual line during this traversal. When a new maximum is found, the variable `\box@thewidth` is updated accordingly. This variable thus contains the maximum line width of all paragraphs within the environment after closing the environment. Traversing the horizontal boxes of a paragraph was inspired by the traversal function described in [3, pages 53–54] and [2].

`box@width` The text within the `\box@width` environment is type-set within a `minipage` environment of width `\linewidth` − `\box@xpos`. The `minipage` environment is required because it enables us to use `\lastbox` (`\lastbox` cannot be used in vertical mode). To prevent that the minipage is put on the current page, we use an `lrbox` to store the minipage.

We define a new `\par` command that performs width calculation by traversing the lines constituting the paragraph. Width calculation is therefore performed automatically after issuing a `\par` command.

Remember from Section A.8 that the macro `\box@currentxpos` uses the value of `\predisplaysize` within a display to calculate the current x position. Remember also that a display ends the current paragraph. LaTeX does not end the paragraph using a `\par` command however. This results in two problems. First, within a display we do not have access to the `\lastbox` of the previous paragraph. The paragraph can therefore not be traversed which makes the width calculation impossible. Secondly, an explicit `\par` command prior to a display sets `\predisplaysize` to zero within the display. The macro `\predisplaysize` is in this case not suitable to calculate the current x position. As a consequence, the implementation of `\box@currentxpos` from Section A.8 cannot be used in combination with the width calculation.

Since we have to traverse all lines of paragraphs for the width calculation, it is rather easy to calculate the width of the last line of a paragraph (which corresponds to the current x position as defined in Section A.8). Within a `minipage` environment no page breaks occur. For the width calculation this is no problem because the text remains invisible after all. Page breaks are required outside the `\box@width` environment however. Calculation of the current x position outside the `\box@width` environment can therefore not performed by traversing the lines of a paragraph.

To be able to calculate the width and the current x position correctly, a re-definition of `\box@xposition` is unavoidable.

The implementation of `\box@width` therefore consists of the re-definition of `\par` and `\box@currentxpos` and the formatting of text in a `minipage` environment of width `\linewidth` − `\box@xpos`.

```
264 \newenvironment{box@width}{%
265 \box@testforboxenv%
```

In order to support nested width calculation, we save the values of some global variables. These are restored after the current width calculation terminates.

```
\edef\box@savedwidth{\the\box@widthincalculation}%
\edef\box@savedlastlinewidth{\the\box@lastlinewidth}%
\edef\box@savedxpos{\the\box@xpos}%
```

We use the value of `\box@leftindent` to accomplish that text following the macro `\box@currentxpos` continuous on the same line. For the width calculation we start formatting text at position 0 (ignoring the left margin) by setting the variables `\box@leftmargin`, `\leftskip`, `\box@xpos`, and `\hangindent` to `\z@`. To construct lines of correct width, we subtract the length of the left margin (`box@leftmargin`) from the line width (`\hsize` and `\linewidth`). Then we start a new paragraph but we continue formatting on the same line. Finally, we enter an `lrbox` environment and we start a `minipage` of width `\linewidth` – `\box@xpos`. The value of `\rightskip` has to be configured again since it is reset when entering the `minipage` environment.

```
\bgroup%
\box@leftindent\box@xpos%
\advance\hsize-\box@leftmargin%
\linewidth\hsize%
\box@leftmargin\z@%
\leftskip\z@%
\box@xpos\z@%
\hangindent\z@%
\par%
\vskip-\parskip%
\begin{lrbox}{\box@widthcalculation}%
\box@widthincalculation=-\maxdimen%
\dimen0=\linewidth%
\advance\dimen0-\box@xpos%
\begin{minipage}{\dimen0}%
\rightskip=0pt plus 1fill%
```

The re-definition of `\par` extends the 'chaining' of `\par` macros. After a new paragraph is started using `\theoldpar`, `\box@traverselines` is called to calculate the width of the previous paragraph by traversing its lines.

```
\def\newpar{%
\theoldpar%
\box@traverselines%
}%
\let\theoldpar=\par%
\let\par=\newpar%
```

During the traversal of the lines of a paragraph, the width of the last line of the paragraph is stored in `\box@lastlinewidth`. The new definition of the macro `\box@currentxpos` first ends the current paragraph and then uses the value of `\box@lastlinewidth` as current x position. When the paragraph was empty, `\box@lastlinewidth` equals $-$`\maxdimen` and we return the left most position (i.e., `\box@savedindentation`).

```
\def\box@currentxpos{%
```

Save the value of `\box@indentation` because it may be changed by `\par`.

```
\edef\savedindentation{\the\box@indentation}%
\par%
\ifdim\box@lastlinewidth=-\maxdimen%
\global\box@xpos\savedindentation%
\else%
\global\box@xpos=\box@lastlinewidth%
\fi%
}%
%\vbox\bgroup%
```

We instantiate `\box@investigateline` such that `\box@thelinewidth` is called by `\box@traverselines` to calculate the maximum line width.

```
\let\box@investigateline\box@thelinewidth%
}{%
```

When closing the `\box@width` environment, we end the current paragraph (to calculate its width) and we close the `minipage` and `lrbox` environments.

```
\par%
\box@traverselines%
\global\box@thewidth=\the\box@widthincalculation%
\end{minipage}%
\end{lrbox}%
```

To support nested width calculations, we restore the values of some global variables.

```
\global\box@xpos=\box@savedxpos%
\global\box@lastlinewidth=\box@savedlastlinewidth%
\global\box@widthincalculation=\box@savedwidth%
\egroup%
}%
```

`box@traverselines` This macro traverses the horizontal boxes of a paragraph. The horizontal boxes are accessed using the `\lastbox` macro. For each horizontal box the macro `\box@investigateline` is called which can be instantiated differently to perform different calculations. Furthermore, this macro stores in `\box@lastlinewidth` the width of the last line (box) of the current paragraph. This variable is used to implement `\box@currentxpos` in the `\box@width` environment.

The traversal and inspection of horizontal boxes of a paragraph using `\lastbox` was inspired by the traversal function described in [3, pages 53–54] and [2] .

```
\newcommand\box@traverselines{%
\global\box@lastlinewidth=-\maxdimen%
```

The traversal function is a recursive function. The recursion is implemented in `\@box@traverselines`. The macro `\box@traverselines` now consists of the initialization of the variable `\box@lastlinewidth` and a single call to `\@box@traverselines`.

```
\def\@box@traverselines{%
\begingroup%
```

We save `\lastbox` in `\box@investigation` and whenever it is a non-void box, we call `\@box@traverselines` recursively, and `\box@investigateline` afterwards. Furthermore, we save the width of the last line in `\box@lastlinewidth`.

```
\setbox\box@investigation=\lastbox%
\ifvoid\box@investigation\else%
\unskip%
\count0=\lastpenalty%
\unpenalty%
\setbox\box@tester=\hbox{\unhcopy\box@investigation}%
```

Calculate the width of the last line.

```
\ifdim\box@lastlinewidth=-\maxdimen%
\global\box@lastlinewidth=\wd\box@tester%
\global\advance\box@lastlinewidth\the\box@indentation%
\fi%
{\@box@traverselines}%
\box@investigateline%
```

After traversing the box, we put it back and restore the penalty that we have removed before.

```
\hbox{\box@indentation\dimen0\unhbox\box@investigation}%
\penalty\count0%
\fi%
\endgroup%
}%
\@box@traverselines%
}%
```

`box@thelinewidth` This macro is used to calculate the maximum width of a sequence of lines. The maximum width so far is stored in `\box@widthincalculation`. The line that is to be inspected is stored in `\box@tester`. Whenever a new maximum has been found (in the case that `\wd\box@tester` + `\box@indentation` > `\box@widthincalculation`), `\box@widthincalculation` is updated.

```
\newcommand{\box@thelinewidth}{%
\begingroup%
\dimen0=\the\wd\box@tester%
\advance\dimen0\box@indentation%
\ifdim\dimen0>\box@widthincalculation%
 \global\box@widthincalculation=\the\dimen0%
\fi%
\endgroup%
}%
```

A.10 Miscellaneous Macros

This section, which concludes the implementation details of `boxenv.sty`, describes the implementation of the remaining macros.

`box@leftindent` This macro inserts horizontal white space to accomplish that text following the macro `\box@leftindent` continues on the current line.

```
\def\box@leftindent#1{%
\aftergroup\insertindent%
```

```
\gdef\insertindent{%
\dimen0=#1%
\advance\dimen0-\box@leftmargin%
\advance\dimen0-\box@indentation%
\makebox [\dimen0]{}%
}%
}
```

box@testforboxenv This macro outputs an error message when the boolean \box@inboxenv yields $false$. The BOX environments should not be used outside a boxenv environment. This macro is used to enforce that the environments are surrounded by a boxenv environment.

```
\newcommand{\box@testforboxenv}{%
\ifbox@inboxenv\else%
\errmessage{Use of environment outside ‘‘boxenv’’
   environment.}%
\fi%
}
```

box@setpenalty We should restrict the number of places in a box@hangpar environment where page breaks may occur. The macro \box@setpenalty is therefore used after each paragraph start to configure the penalties.

We allow a page break between two paragraphs within in a VBOX or HVBOX environment to occur only when their vertical space factor (the vs space option) is greater than zero. No page breaks are allowed in other environments (except the latextext environment). Since text within latextext is formatted as ordinary text, penalties are restored to their default values within this environment.

```
\newcommand{\box@setpenalty}{%
```

When \parskip equals zero (i.e., the is space option equals zero), we disable page breaking by setting \penalty to 10000. Otherwise, a default penalty of 50 is used.

```
\ifdim\parskip=\z@%
\penalty10000%
\else%
\penalty 50%
\fi%
```

We always disable page breaks between lines and after discretionary hyphens.

```
\linepenalty10000%
\interlinepenalty=10000%
\hyphenpenalty=10000%
}%
```

box@wcopies The macro \box@wcopies constructs a line of width approximately equal to #1. The exact width of the line depends on #2:

- When #2 equals the symbol ‘=’, a line is constructed using \hrule. Its width equals #1.

- Otherwise, a line is constructed by taking n copies of #2 such that: $n \times |\#2| \leq \#1 < (n+1) \times |\#2|$.

```
\newcommand{\box@wcopies}[2]{%
\if#2=%
\vskip-1.5\baselineskip%
\leavevmode\hbox to #1 {\leaders\hrule\hfill}%
\else%
 \newbox\tmp%
 \setbox\tmp=\hbox{}%
 \loop\ifdim#1>\wd\tmp%
  \setbox\tmp\hbox{#2\box\tmp}%
 \repeat%
 \hbox{\hbox to \box@leftmargin{\box\tmp}}%
\fi%
}%
```

box@absval `\box@absval` is a little macro that expands to the absolute value of its argument.

```
\def\box@absval#1{\ifnum#1<\z@ -\fi#1}%
```

ifundefined This macro checks whether its argument has been previously defined in the document. It has been taken from [4, page 40, page 308].

```
\def\ifundefined#1{%
\expandafter\ifx\csname #1\endcsname%
\relax%
}
```