marginalia — Non-floating marginal content with automatic placement for LuaLaTeX[*]

Alan J. Cain

Released 2025-09-11

Abstract

This LuaLaTeX package allows the placement of marginal content anywhere, without `\marginpar`'s limits, and automatically adjusts positions to prevent overlaps or content being pushed off the page. In short, it tries to combine the best features from the packages marginnote, marginfix and marginfit with key–value settings that allow fine-grained customization.

Contents

[*]This file describes v0.81.5, last revised 2025-09-11.

Requirements

1 Introduction

The LaTeX `\marginpar` command is the basic method for placing content in the margin. For purposes such as drawing attention to particular points in the text, it functions well. Its main limitation is that `\marginpar` works via the LaTeX float mechanism and so cannot be used to create marginal content next to a figure, table, or other float, or next to a footnote, or to place running heads in the margin, such as are found in the left-hand margin of this document except for the 'implementation' section. (Bringhurst called this style 'running shoulderheads' [Bri04, p. 65], but the term may be non-standard.)

Trying to set many separate pieces of marginal content using `\marginpar` can lead to other problems. If two `marginpar`s would clash, LaTeX shifts the second item downward. But the cumulative effect can lead to `marginpar`s being shifted downward off the bottom of the page. Further, the asynchronous nature of TeX's page-breaking can cause: (1) a `marginpar` to be placed in the wrong margin; (2) the topmost `marginpar` on a page to be unnecessarily shifted downward because of a hypothetical clash that would have occured with the previous `marginpar`, had they been on the same page.

Packages like mparhack[1] (Tom Sgouros & Stefan Ulrich), marginnote[2] (Markus Kohm), marginfix[3] (Stephen Hicks) and marginfit[4] (Maurice Leclaire) were created to avoid these limitations and problems. mparhack only ensures that each `marginpar` appears on the correct side of the page. marginnote allows marginal content to be placed anywhere, but does not adjust positions to avoid clashes. marginfix adjusts positions, but the unadjusted vertical positioning can be slightly off, and the package still uses floats. marginfit gets positions exactly right, but uses the insert mechanism and so marginal content cannot appear next to floats or footnotes.

1 URL: https://ctan.org/pkg/mparhack
2 URL: https://ctan.org/pkg/marginnote
3 URL: https://ctan.org/pkg/marginfix
4 URL: https://ctan.org/pkg/marginfit

This LuaLaTeX package, marginalia, provides a `\marginalia` command that attempts to avoid these limitations. Marginal content is placed, not via floats or inserts, but by a calculated per-item horizontal shift inside an (invisible) `\rlap` or `\llap` from the position where the `\marginalia` command was issued (which is similar to the technique used by marginnote), plus a calculated per-item vertical shift to avoid clashes with other content. The vertical shift is usually downward, but may be upward when necessary to prevent content from being shifted off the bottom of the page (which is similar to the vertical shifts performed by marginfix and marginfit).

The calculation of the horizontal and vertical shifts uses information written to the `.aux` file during the previous LuaLaTeX run. It thus takes at least two runs for all content to appear in the correct places. The package reports any changes from the previous run and any problems encountered.

Note: marginalia was written to typeset running heads in the margin, sidenote references, side-captions for floats, and small marginal figures in the author's book *Form & Number: A History of Mathematical Beauty* [Cai24].[5] Thus the basic functionality has been tested extensively, and it has performed correctly.

5 *Form & Number* is freely available on the Internet Archive under a Creative Commons licence. URL: https://archive.org/details/cain_formandnumber_ebook_large

Acknowledgements. The author thanks Ulrike Fischer for explaining how to add tagging support.

[1] URL: https://www.latex-project.org/lppl.txt

2 Requirements

marginalia requires

(1) LuaLaTeX,

(2) a recent LaTeX kernel with expl3 support (any kernel version since 2020-02-02 should suffice).

It does not depend on any other packages.

3 Installation

To install marginalia manually, run `luatex marginalia.ins` and copy `marginalia.sty` and `marginalia.lua` to somewhere LuaLaTeX can find them.

4 Getting started

marginalia works 'out of the box'. Load the package (there are no package options) and use the main `\marginalia` command to place marginal content. Figure 4.1 shows the source code for a small demonstration and the resulting document. *The source code must be processed twice by LuaLaTeX for the marginal content to be placed correctly.* (See Section 8 for discussion of the need for multiple runs.)

Turn to Section 5 for a detailed description of the available user commands, and Section 6 for the various options (such as `style=⟨code⟩`) than can be used to change the placement and formatting of the marginal content.

5 User commands

`\marginalia` `\marginalia[⟨options⟩]{⟨content⟩}`

This is the basic command for placing marginal content. The ⟨*content*⟩ can, roughly speaking, be anything: text, mathematics, included graphics, Ti*k*Z. The optional argument ⟨*options*⟩ is a key–value list that specifies how the content is typeset. The keys are described in Subsection 6.

`\marginaliasetup` `\marginaliasetup{⟨options⟩}`

This command is used to set options for all subsequent calls to `\marginalia`. The argument ⟨*options*⟩ is the same kind of key–value list as the ⟨*options*⟩ argument for the `\marginalia` command, and the keys are described in Subsection 6.

Note that `\marginaliasetup` can be used in the preamble or in the body of the document.

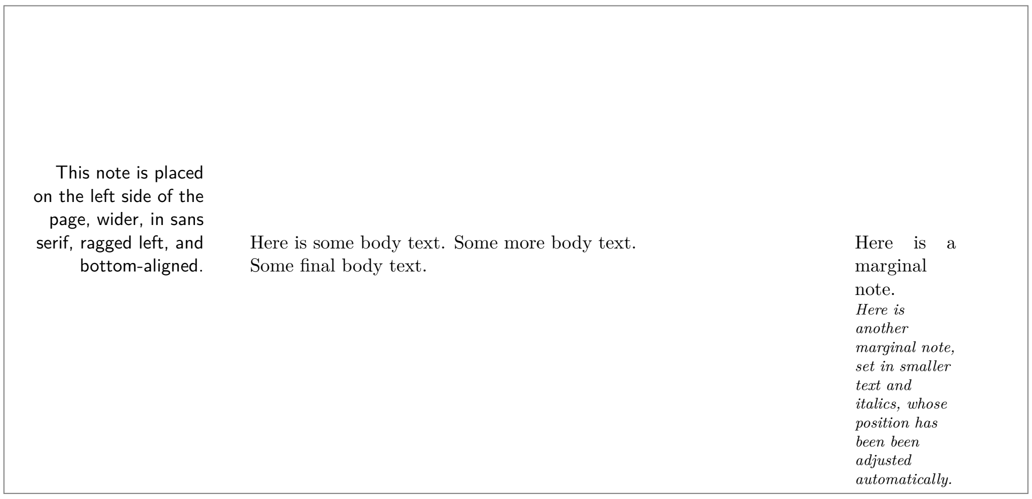
`\marginalianewgeometry` `\marginalianewgeometry`

This command signals to marginalia that the page layout has been changed, for instance by using the `\newgeometry` command from the geometry package,[6] or by using the LaTeX command `\twocolumn` to switch to two-column mode. It should be issued immediately after such a change, and certainly before the first page with the new layout has been shipped out. There is no harm in using it unnecessarily.

6 URL: https://ctan.org/pkg/geometry

User commands

```
\documentclass[11pt]{article}

\usepackage{marginalia}

\begin{document}

Here is some body text.\marginalia{Here is a marginal note.} Some more
body text.\marginalia[style=\footnotesize\itshape\raggedright]{Here is another
  marginal note, set in smaller text and italics, whose position has been been
  adjusted automatically.}

Some final body text.\marginalia[pos=left, valign=b, style=\sffamily\raggedleft,
width=35mm]{This note is placed on the left side of the page, wider, in sans
  serif, ragged left, and bottom-aligned.}

\end{document}
```



This note is placed on the left side of the page, wider, in sans serif, ragged left, and bottom-aligned.

Here is some body text. Some more body text. Some final body text.

Here is a marginal note.

Here is another marginal note, set in smaller text and italics, whose position has been been adjusted automatically.

Figure 4.1: A small demonstration of marginalia.

5.1 Access to page and column

Within the ⟨*content*⟩ of `\marginalia`, two counters are available which specify the actual page and column in which the call to `\marginalia` appears. These counters can be used to select different actions depending on the page on which the content appears or (in two-column mode) whether it pertains to the left or right column. It is best to use the variants of the `style` and `width` keys if marginal content should have different widths or styles depending on whether they appear on a recto/verso page or pertain to a particular column. These counters are made available for purposes not covered by the `style` and `width` variants. The value of each counter is based on the position of the call to `\marginalia` on the previous LuaLaTeX run.

`\marginaliapage` A counter register, available within the ⟨*content*⟩ of `\marginalia`, that holds the actual page on which the marginal content appears. The value is based on the previous LuaLaTeX run and will default to 1.

`\marginaliacolumn` A counter register, available within the ⟨*content*⟩ of `\marginalia`, that holds the actual column to which the marginal content pertains. The value is 1 for the left column, 2 for the right column. In one-column mode, the value is always 0. (If the key `column` is used to manually specify the column to which the content pertains, the value of `\marginaliacolumn` will change accordingly.) The value is based on the previous LuaLaTeX run and will default to 0.

6 Options

The description of keys in this section, which are summarized in Table 1, should be read in conjunction with the discussion of how marginal content is placed in Section 7. In particular, the variants of the keys `width` and `style` follow the terminology shown in Figure 7.1.

6.1 Type

`type` The `type` of an item of marginal content can be set to one of the following three values:

`normal`: The vertical position of the item will be changed automatically if necessary to prevent a clash with another item of content.

`fixed`: The vertical position of the item will *never* be changed automatically from the position specified by `yshift`, even if there is a clash with another item. (The type `fixed` was designed for setting float captions in the margin, since a caption should not move away from the float with which it is associated.)

`optfixed`: The vertical position of the item will *never* be changed automatically from the position specified by `yshift`, even if there is a clash with another item. But an `optfixed` item will not appear in the document if it would clash with a `fixed` item. (The type `optfixed` was designed for setting running heads in the margin, which should not appear if they would clash with a figure caption set in the margin.)

(*Default:* `normal`)

6.2 Horizontal placement

`pos` The position in which an item of marginal content should be placed. It can be set to one of the the following four values:

`auto`: Place the item in the default position as described in Section 7: the outer margin in single-column mode, and on the opposite side from the other column in two-column mode.

`reverse`: Place the item on the opposite side of the text block (in one-column mode) or column (in two-column mode) from `auto`.

`left`: The left side of the text block or column.

`right`: The right side of the text block of column.

`nearest`: The side of the text block or column nearest to which `\marginalia` was called.

(*Default:* `auto`)

Options

Table 1: Summary of keys that can be set using `\marginaliasetup` or passed in the optional argument to `\marginalia`.

Key name	Value	Default
`type`	{`normal`, `fixed`, `optfixed`}	`normal`
`pos`	{`auto`, `reverse`, `left`, `right`, `nearest`}	`auto`
`column`	{`auto`, `one`, `left`, `right`}	`auto`
`xsep`	Dimension	`\marginparwidth`
`xsep outer`	Dimension	`\marginparwidth`
`xsep inner`	Dimension	`\marginparwidth`
`xsep between`	Dimension	`\marginparwidth`
`xsep recto outer`	Dimension	`\marginparwidth`
`xsep recto inner`	Dimension	`\marginparwidth`
`xsep verso outer`	Dimension	`\marginparwidth`
`xsep verso inner`	Dimension	`\marginparwidth`
`xsep right between`	Dimension	`\marginparwidth`
`xsep left between`	Dimension	`\marginparwidth`
`valign`	{`t`, `b`}	`t`
`yshift`	Dimension	0 pt
`ysep`	Dimension	`\marginparpush`
`ysep above`	Dimension	`\marginparpush`
`ysep below`	Dimension	`\marginparpush`
`ysep page top`	Dimension	`\marginparpush`
`ysep page bottom`	Dimension	`\marginparpush`
`width`	Dimension	`\marginparwidth`
`width outer`	Dimension	`\marginparwidth`
`width inner`	Dimension	`\marginparwidth`
`width between`	Dimension	`\marginparwidth`
`width recto outer`	Dimension	`\marginparwidth`
`width recto inner`	Dimension	`\marginparwidth`
`width verso outer`	Dimension	`\marginparwidth`
`width verso inner`	Dimension	`\marginparwidth`
`width right between`	Dimension	`\marginparwidth`
`width left between`	Dimension	`\marginparwidth`
`style`	LaTeX code	[Empty]
`style recto outer`	LaTeX code	[Empty]
`style recto inner`	LaTeX code	[Empty]
`style verso outer`	LaTeX code	[Empty]
`style verso inner`	LaTeX code	[Empty]
`style right between`	LaTeX code	[Empty]
`style left between`	LaTeX code	[Empty]

Options

`column` In two-column mode, `marginalia` tries to determine to which column an item of marginal content pertains using the position of the call to `\marginalia`. If the call is to the left of the mid-point between the columns, the item is assumed to pertain to the left column; otherwise, it is assumed to pertain to the right column. In certain situations, this might lead to undesired placement of the item. In particular, any call to `\marginalia` in a full-width float in two-column mode would be handled as if it were a call from one of the columns and might thus be set in the wrong place. Similarly, an overfull hbox or a piece of `\rlap`-ped text might carry a call to `\marginalia` from the left column text into the area of the page occupied by the right column.

The key `column` can be used to specify which column `marginalia` should place the item in. It can be set to one of four values:

`auto`: Automatically determine which column an item of marginal content is placed in.

`one`: Treat the item as being called from one-column mode.

`left`: Treat the item as pertaining to the left column.

`right`: Treat the item as pertaining to the right column.

The value of `column` has no effect in one-column mode. (*Default:* `auto`)

`xsep`
`xsep outer`
`xsep inner`
`xsep between`
`xsep recto outer`
`xsep recto inner`
`xsep verso outer`
`xsep verso inner`
`xsep right between`
`xsep left between`

These keys specify the horizontal separation between an item of marginal content and the text block next to which it is placed. Which separation is used will depend on where the item is typeset. The terminology is as in Figure 7.1.

`xsep recto outer`: used for an item in the outer margin of a recto page.

`xsep recto inner`: used for an item in the inner margin of a recto page.

`xsep verso outer`: used for an item in the outer margin of a verso page.

`xsep verso inner`: used for an item in the inner margin of a verso page.

`xsep right between`: used for an item set from the right column between the columns.

`xsep left between`: used for an item set from the left column between the columns.

`xsep outer`: a shorthand for setting the keys `xsep recto outer` and `xsep verso outer` simultaneously to the same value.

`xsep inner`: a shorthand for setting the keys `xsep recto inner` and `xsep verso inner` simultaneously to the same value.

`xsep between`: a shorthand for setting the keys `xsep right between` and `xsep left between` simultaneously to the same value.

`xsep`: a shorthand for setting all of these keys simultaneously.

(The shorthands `xsep outer` and `xsep inner` exist because page geometry is usually symmetrical between recto and verso pages as regards outer and inner margins. The shorthand `xsep between` exists because the space between columns, if used at all for marginal content, will often be shared equally.) Each of these keys must be set to a valid dimension. (*Default:* value of `\marginparsep` when the package is loaded)

6.3 Vertical placement

`valign` The option `valign` can be either `t` or `b`. In the former case, the baseline of the marginal content item is the baseline of the topmost box in its contents; in the latter case, its baseline is the baseline of the bottommost box in its contents. (Essentially, `\vtop` and `\vbox` are used to set the two options) (*Default:* `t`)

Options

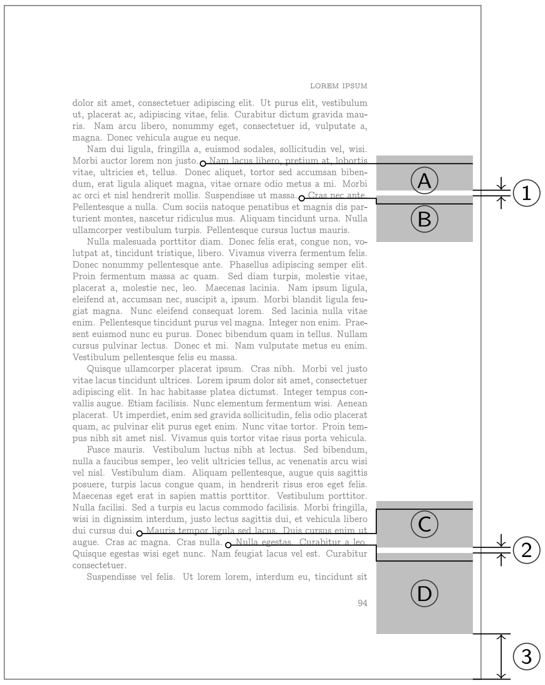


Figure 6.1: (Illustration of `ysep`) The length ① is at least the value of `ysep below` specified (locally or globally) for marginal content item Ⓐ and at least the value of `ysep above` specified for item Ⓑ. In this example diagram, Ⓑ has been automatically moved down from its natural position to maintain the required distance. Similarly, the length ② is at least the value of `ysep below` specified for Ⓒ and at least the value of `ysep above` specified for Ⓓ, and the length ③ is at least the value of `ysep page bottom` specified for Ⓓ. In this example, to maintain the required distances, Ⓒ and Ⓓ have been automatically moved (respectively) up and down from their natural positions.

`yshift` The key `yshift` is used to shift the default position of the marginal content item up (positive) or down (negative) from its normal position, which is to have its baseline aligned with the baseline of the callout position. It must be set to a valid dimension. Note that if `type=normal`, then the vertical position may be adjusted from that specified by `yshift`. If this is not desired, specify a different `type`. (*Default:* 0pt).

`ysep`
`ysep above`
`ysep below`
`ysep page top`
`ysep page bottom`

These keys specify the minimum vertical separation above and below an item of marginal content

`ysep above:` the minimum vertical separation between an item and the one above.

`ysep below:` the minimum vertical separation between an item and the one below.

`ysep page top:` the minimum vertical separation between an item and top of the page.

`ysep page bottom:` the minimum vertical separation between an item and bottom of the page.

`ysep:` is a shorthand for setting all of these keys simultaneously to the same value.

(See Figure 6.1.) Each of these keys must be set to a valid dimension. (*Default:* value of `\marginparpush` when the package is loaded).

6.4 Appearance

An item of marginal content that appears in the inner margin might be narrower than one that appears in the outer margin, and an item appearing in the outer margin of a recto page might be set ragged right, while an item appearing in the outer margin of a verso page might be set ragged left. And since it is not known where an item will appear until the page is assembled, the keys in this subsection, dealing with the width and style of an item, have variants that apply depending on where the item appears on the page.

`width`
`width outer`
`width inner`
`width between`
`width recto outer`
`width recto inner`
`width verso outer`
`width verso inner`
`width right between`
`width left between`

These keys specify the width of the an item of marginal content (or, more precisely, the `\hsize` of the box into which the item is typeset). Which width is chosen will depend on the where the item is typeset. The terminology is as in Figure 7.1.

- **`width recto outer:`** used for an item in the outer margin of a recto page.
- **`width recto inner:`** used for an item in the inner margin of a recto page.
- **`width verso outer:`** used for an item in the outer margin of a verso page.
- **`width verso inner:`** used for an item in the inner margin of a verso page.
- **`width right between:`** used for an item set from the right column and placed between the columns.
- **`width left between:`** used for an item set from the right column and placed between the columns.
- **`width outer:`** a shorthand for setting the keys `width recto outer` and `width verso outer` simultaneously to the same value.
- **`width inner:`** a shorthand for setting the keys `width recto inner` and `width verso inner` simultaneously to the same value.
- **`width between:`** a shorthand for setting the keys `width right between` and `width left between` simultaneously to the same value.
- **`width:`** a shorthand for setting all of these keys simultaneously.

(The shorthands `width outer` and `width inner` exist because page geometry is usually symmetrical between recto and verso pages as regards outer and inner margins. The shorthand `width between` exists because the space between columns, if used at all for marginal content, will often be shared equally.) Each of these keys must be set to a valid dimension. (*Default:* value of `\marginparwidth` when the package is loaded)

`style`
`style recto outer`
`style recto inner`
`style verso outer`
`style verso inner`
`style right between`
`style left between`

These keys specify the style with which an item of marginal content is typeset. Which style is chosen will depend on where the item is typeset. The terminology is as in Figure 7.1.

- **`style recto outer:`** used for an item in the outer margin of a recto page.
- **`style recto inner:`** used for an item in the inner margin of a recto page.
- **`style verso outer:`** used for an item in the outer margin of a verso page.
- **`style verso inner:`** used for an item in the inner margin of a verso page.
- **`style right between:`** used for an item set from the right column between the columns.
- **`style left between:`** used for an item set from the right column between the columns.
- **`style:`** a shorthand for setting all of these keys simultaneously.

Each of these keys should be set to LaTeX code that specifies the style. (*Default:* [Empty])

7 Placement

The placement of an item of marginal content depends on where the call to `\marginalia` appears in the finished document. Both horizontal and vertical placement can be com-

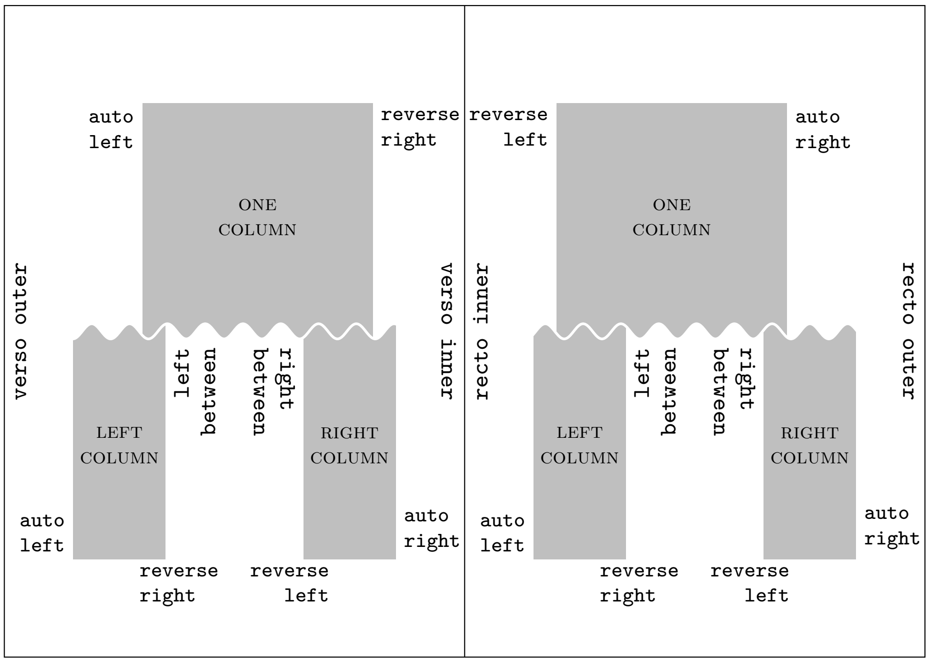


Figure 7.1: Summary of the positioning of marginal content using `pos`, and terminology used in `width` and `style` keys, on recto and verso pages, in both one-column and two-column mode.

plicated.

7.1 Horizontal placement

To understand the horizontal placement, first recall some terminology: a recto page is an odd-numbered page in two-sided mode, or any page in one-sided mode; a verso page is an even-numbered page in two-sided mode. The description in the paragraphs that follow is summarized in Figure 7.1.

In one-column mode, marginal content is placed by default in the outer margin: right on recto pages, left on verso pages. If `pos=reverse` is applied, it is placed in the inner margin: left on recto pages, right on verso pages.

In two-column mode, the default placement is next to the column in which the call to `\marginalia` appears, on the side opposite to the other column. Thus, if the call to `\marginalia` was in the left column, the marginal content item is placed by default on the left: on a recto page, the inner margin, on a verso page, the outer margin. If `pos=reverse` is applied, it is placed between the two columns, adjacent to the left column. If the call to `\marginalia` was in the right column, the item is placed by default on the right: on a recto page, the outer margin, on a verso page, the inner margin. If `pos=reverse` is applied, it is placed between the two columns, adjacent to the right column.

`pos=left` specifies that the item is to be placed on the left of the text block or column containing the call to `\marginalia`.

`pos=right` similarly specifies that the item is to be placed on the right of the text block or column containing the call to `\marginalia`.

Usage notes marginalia determines in which column the call to `\marginalia` was made using its horizontal position. As discussed in the description of key `column`, there are situations where this can go wrong and which necessitate a manual specification of a particular column.

7.2 Vertical placement

marginalia tries by default to place the each item of marginal content with its baseline shifted by the value of `yshift` (by default, 0 pt) from the baseline where `\marginalia` was called. The actual vertical placement is calculated by the procedure described below, carried out for the items appearing in a particular horizontal location. (As shown in Figure 7.1, in one-column mode the possible locations are in outer and inner margins; in two-column mode the possible locationd are the outer and inner margins and on the left and right sides of the space between the columns.) A *clash* exists when two items are closer than specified by `ysep below` for the upper item or `ysep above` for the lower item, whichever is greater.

For the items in each horizontal location, the procedure is as follows:

1. Place the items appearing in a given horizontal location on the page into a list.
2. Set the vertical shift of each item to the one specified by `yshift`.
3. For each `type=optfixed` item, if it clashes with any `type=fixed` item, delete it from the list of items that appear on the page.
4. Sort the list by the position of the call to `\marginalia`, top-to-bottom, left-to-right, breaking ties by the order of calls. (Because of floats, footnotes, etc., the sorted order of the list is not necessarily the same as the order of appearance of `\marginalia` commands in the source code.)
5. Pass through the list of items in sorted order. For each `type=normal` item, if necessary shift it in a negative (downward) direction so that it (1) does not reach closer to the top of the page than specified by `ysep page top`, and (2) does not clash with the previous (above) item. (After this stage, it is possible for an assigned vertical shift to push a `type=normal` item off the bottom of the page.)
6. Pass through the list of items in the reverse of the sorted order. For each `type=normal` item, if necessary shift it in a positive (upward) direction so that it (1) does not reach closer to the bottom of the page than specified by `ysep page bottom`, and (2) does not clash with the next (below) item.

During this process, it may be found that it is impossible to prevent clashes or items reaching beyond the limits (e.g. fixed items clash with each other; a fixed item conflicts with `ysep page top` or `ysep page bottom`, or there are simply too many items of marginal content to fit (in which case, the top of some of them will be above the limit specified by `ysep page top` or will clash with fixed items)). In these cases, warnings are issued at the end of the LuaLaTeX run.

8 Usage notes

marginalia requires a minimum of two LuaLaTeX runs, and often more, to place items of marginal content correctly. On the first pass, information about items, including their

Limitations

vertical size, is written to the `.aux` file, and this information is used to position them correctly on the next run. However, because `width` and `style` have variants dependent on the margin in which the item is placed, an item may only be typeset at the correct size on this second run. Thus the vertical size of the item may have changed and so the information written to the `.aux` file on the previous run may be out of date. In this case a third run may be needed for correct placement.

More runs may be needed if the position of the call to `\marginalia` changes between runs. Provided the main text stabilizes, the placement of items using `\marginalia` should be correct two runs later.

At the end of the LuaLaTeX run, marginalia reports any problems encountered in the vertical placement of items (as decribed at the end of Subsection 7.2). These problems are based on calculations made on the basis of information from the previous written to the `.aux` file on the previous run, and may not arise if item positions or sizes (i.e. height or depth) have changed. marginalia also reports any changes in positions or sizes compared to the previous run.

In these reports, a page number refers to a visible page number if it is prefixed with '`p`'; it otherwise refers to the absolute page number of the output.

9 Incompatibilities

Using marginalia alongside `\marginpar` or packages like mparhak, marginnote, marginfix, or marginfit should not produce any errors, but marginalia will ignore marginal content not created using `\marginalia`; for example, an item of marginal content created using `\marginalia` might overlap with one created using `\marginpar`.

10 Limitations

As noted in the introduction, marginalia was originally written to typeset a particular kind of book. It thus has several limitations. Three of these are:

LuaLaTeXonly Most of the code for deciding the placement of items of marginal content is written in Lua. In principle, it could be replaced with a pure LaTeX solution.

No support for 'moving past' fixed items The adjustment of vertical positions will never cause a `type=normal` item to be shifted past a `type=fixed` one, even when there is space on the other side. It may be desirable to have this available as an option.

No support for nested content items Nesting might be desirable for typesetting editions of manuscripts which sometimes contain marginal glosses, and then glosses upon those glosses.

The lack of any built-in facility for producing (for example) numbered sidenotes is a conscious design choice. This is properly the concern of a command that merely uses `\marginalia` to place the notes correctly.

References

[Bri04] R. Bringhurst. *The Elements of Typographic Style*. Hartley & Marks, version 3.0, 2004.

[Cai24] A. J. Cain. *Form & Number: A History of Mathematical Beauty*. Lisbon, 2024. URL: https://archive.org/details/cain_formandnumber_ebook_large.

11 Implementation (LaTeX package)

```
⟨*package⟩
⟨@@=marginalia⟩
```

11.1 Initial set-up

Package identification/version information.

```
\NeedsTeXFormat{LaTeX2e}[2020-02-02]
\ProvidesExplPackage{marginalia}{2025-09-11}{0.81.5}
  {Non-floating marginal content for LuaLaTeX}
```

Check that LuaTeX is in use.

```
\sys_if_engine_luatex:F
  {
    \msg_new:nnn{marginalia}{lualatex_required}
      {LuaLaTeX~required.~Package~loading~will~abort.}
    \msg_critical:nn{marginalia}{lualatex_required}
  }
```

11.2 Tagging set-up

If LaTeX has tagging support, set up sockets if necessary and define `\__marginalia_-tagging_socket:n` to be `\UseTaggingSocket`.

```
\@ifundefined{UseTaggingSocket}
  {
    \cs_new:Npn \__marginalia_tagging_socket:n #1 {}
  }
  {
    \str_if_exist:cF { l__socket_tagsupport/marginpar/begin_plug_str }
      {
        \socket_new:nn {tagsupport/marginpar/begin}{0}
        \socket_new:nn {tagsupport/marginpar/end}{0}
      }
    \str_if_exist:cF { l__socket_tagsupport/para/restore_plug_str }
      {
        \socket_new:nn {tagsupport/para/restore}{0}
      }
    \cs_new:Npn \__marginalia_tagging_socket:n #1
      {
        \UseTaggingSocket{#1}
      }
  }
```

11.3 Options

Set up the key–value options and the variables in which the settings will be stored.

11.3.1 Type

\l__marginalia_type_int

A key to store the type of the marginal content item. The setting is held in an integer variable: 1 = `normal`, 2 = `fixed`, 3 = `optfixed`.

```
\int_new:N\l__marginalia_type_int
\keys_define:nn { marginalia }
{
  type .choices:nn = {normal,fixed,optfixed}{
    \int_set:Nn\l__marginalia_type_int{\l_keys_choice_int}
  },
  type .initial:n = normal,
}
```

(*End of definition for* `\l__marginalia_type_int`*.*)

11.3.2 Horizontal placement

\l__marginalia_pos_int

A key to store the specified position of the marginal content item. The setting is held in an integer variable: 1 = `auto`, (the outer margin in one-column mode; left margin in left column, right margin in right column in two-column mode) 2 = `reverse` (inner margin in one-column mode; between the columns in two-column mode), 3 = `left`, 4 = `right`, 5 = `nearest`.

```
\int_new:N\l__marginalia_pos_int
\keys_define:nn { marginalia }
{
  pos .choices:nn = {auto,reverse,left,right,nearest}{
    \int_set:Nn\l__marginalia_pos_int{\l_keys_choice_int}
  },
  pos .initial:n = auto
}
```

(*End of definition for* `\l__marginalia_pos_int`*.*)

\l__marginalia_column_int

A key to force the marginal content item to be treated in one-column mode or as being set from the left or right column. The setting is held in an integer variable: −1 = `auto` (automatic), 0 = `one` (one-column mode), 1 = `left` (left column) 2 = `right` (right column).

```
\int_new:N\l__marginalia_column_int
\keys_define:nn { marginalia }
{
  column .choices:nn = {auto,one,left,right}{
    \int_set:Nn\l__marginalia_column_int{\l_keys_choice_int-2}
  },
  column .initial:n = auto,
}
```

(*End of definition for* `\l__marginalia_column_int`*.*)

\l__marginalia_xsep_recto_outer_dim
\l__marginalia_xsep_recto_inner_dim
\l__marginalia_xsep_verso_outer_dim
\l__marginalia_xsep_verso_inner_dim
\l__marginalia_xsep_right_between_dim
\l__marginalia_xsep_left_between_dim

Dimension keys to hold the separation between the marginal content item and the main text, which can be dependent on where it appears on the page.

```
\keys_define:nn { marginalia }
{
  xsep~recto~outer .dim_set:N = \l__marginalia_xsep_recto_outer_dim,
  xsep~recto~inner .dim_set:N = \l__marginalia_xsep_recto_inner_dim,
  xsep~verso~outer .dim_set:N = \l__marginalia_xsep_verso_outer_dim,
  xsep~verso~inner .dim_set:N = \l__marginalia_xsep_verso_inner_dim,
  xsep~right~between .dim_set:N = \l__marginalia_xsep_right_between_dim,
  xsep~left~between .dim_set:N = \l__marginalia_xsep_left_between_dim,
  xsep .code:n = {
    \keys_set:nn{ marginalia }{
      xsep~recto~outer=#1,
      xsep~recto~inner=#1,
      xsep~verso~outer=#1,
      xsep~verso~inner=#1,
      xsep~right~between=#1,
      xsep~left~between=#1,
    }
  },
  xsep~outer .code:n = {
    \keys_set:nn{ marginalia }{
      xsep~recto~outer=#1,
      xsep~verso~outer=#1,
    }
  },
  xsep~inner .code:n = {
    \keys_set:nn{ marginalia }{
      xsep~recto~inner=#1,
      xsep~verso~inner=#1,
    }
  },
  xsep~between .code:n = {
    \keys_set:nn{ marginalia }{
      xsep~right~between=#1,
      xsep~left~between=#1,
    }
  },
  xsep .initial:n = \marginparsep,
}
```

(*End of definition for* `\l__marginalia_xsep_recto_outer_dim` *and others.*)

11.3.3 Vertical placement

`\l__marginalia_valign_int`

A key to store the vertical alignment of the marginal content item. The setting is held in a integer variable: $1 =$ `t` (aligned at the baseline of the topmost line of the item), $2 =$ `b` (aligned at the baseline of the bottommost line of the item).

```
\int_new:N\l__marginalia_valign_int
\keys_define:nn { marginalia }
{
  valign .choices:nn = {t,b}{
    \int_set_eq:NN\l__marginalia_valign_int\l_keys_choice_int
  },
  valign .initial:n = t,
}
```

(*End of definition for* `\l__marginalia_valign_int`*.*)

`\l__marginalia_default_yshift_dim`

Dimension key to hold the default vertical shift of the marginal content item from its natural position.

```
\keys_define:nn { marginalia }
{
  yshift .dim_set:N = \l__marginalia_default_yshift_dim,
  yshift .initial:n = 0pt,
}
```

(*End of definition for* `\l__marginalia_default_yshift_dim`*.*)

`\l__marginalia_ysep_above_dim`
`\l__marginalia_ysep_below_dim`
`\l__marginalia_ysep_page_top_dim`
`\l__marginalia_ysep_page_bottom_dim`

Dimension keys to hold the the minimum vertical spacing between a marginal content item and (respectively) the item above, the item below, the page top, and the page bottom.

```
\keys_define:nn { marginalia }
{
  ysep~above .dim_set:N = \l__marginalia_ysep_above_dim,
  ysep~below .dim_set:N = \l__marginalia_ysep_below_dim,
  ysep~page~top .dim_set:N = \l__marginalia_ysep_page_top_dim,
  ysep~page~bottom .dim_set:N = \l__marginalia_ysep_page_bottom_dim,
  ysep .code:n = {
    \keys_set:nn{ marginalia }{
      ysep~below=#1,
      ysep~above=#1,
      ysep~page~top=#1,
      ysep~page~bottom=#1,
    }
  },
  ysep .initial:n = \marginparpush,
}
```

(*End of definition for* `\l__marginalia_ysep_above_dim` *and others.*)

11.3.4 Appearance

`\l__marginalia_width_recto_outer_dim`
`\l__marginalia_width_recto_inner_dim`
`\l__marginalia_width_verso_outer_dim`
`\l__marginalia_width_verso_inner_dim`
`\l__marginalia_width_right_between_dim`
`\l__marginalia_width_left_between_dim`

Dimension keys to hold the width of the marginal content item, which can be dependent on where it appears on the page.

```
\keys_define:nn { marginalia }
{
  width~recto~outer .dim_set:N = \l__marginalia_width_recto_outer_dim,
  width~recto~inner .dim_set:N = \l__marginalia_width_recto_inner_dim,
  width~verso~outer .dim_set:N = \l__marginalia_width_verso_outer_dim,
  width~verso~inner .dim_set:N = \l__marginalia_width_verso_inner_dim,
  width~right~between .dim_set:N = \l__marginalia_width_right_between_dim,
  width~left~between .dim_set:N = \l__marginalia_width_left_between_dim,
  width .code:n = {
    \keys_set:nn{ marginalia }{
      width~recto~outer=#1,
      width~recto~inner=#1,
      width~verso~outer=#1,
      width~verso~inner=#1,
      width~right~between=#1,
      width~left~between=#1,
    }
  },
  width~outer .code:n = {
    \keys_set:nn{ marginalia }{
      width~recto~outer=#1,
      width~verso~outer=#1,
    }
  },
  width~inner .code:n = {
    \keys_set:nn{ marginalia }{
      width~recto~inner=#1,
      width~verso~inner=#1,
    }
  },
  width~between .code:n = {
    \keys_set:nn{ marginalia }{
      width~right~between=#1,
      width~left~between=#1,
    }
  },
  width .initial:n = \marginparwidth,
}
```

(*End of definition for* `\l__marginalia_width_recto_outer_dim` *and others.*)

\l__marginalia_style_recto_outer_tl
\l__marginalia_style_recto_inner_tl
\l__marginalia_style_verso_outer_tl
\l__marginalia_style_verso_inner_tl
\l__marginalia_style_right_between_tl
\l__marginalia_style_left_between_tl

Token list keys to hold the style with which a marginal content item is typeset, which can be dependent on where it appears on the page.

```
\keys_define:nn { marginalia }
{
  style~recto~outer .tl_set:N = \l__marginalia_style_recto_outer_tl,
  style~recto~inner .tl_set:N = \l__marginalia_style_recto_inner_tl,
  style~verso~outer .tl_set:N = \l__marginalia_style_verso_outer_tl,
  style~verso~inner .tl_set:N = \l__marginalia_style_verso_inner_tl,
  style~right~between .tl_set:N = \l__marginalia_style_right_between_tl,
  style~left~between .tl_set:N = \l__marginalia_style_left_between_tl,
  style .code:n = {
    \keys_set:nn{ marginalia }{
      style~recto~outer=#1,
      style~recto~inner=#1,
      style~verso~outer=#1,
      style~verso~inner=#1,
      style~right~between=#1,
      style~left~between=#1,
    }
  },
  style .initial:n = {},
}
```

(*End of definition for* `\l__marginalia_style_recto_outer_tl` *and others.*)

11.4 Lua backend and interface

Load the Lua backend.

```
  \lua_now:n{
    marginalia = require('marginalia')
  }
```

The following 9 macros interface between LaTeX and Lua code. Each control sequence `\__marginalia_lua_XYZ` simply calls the corresponding Lua function `marginalia.XYZ`.

__marginalia_lua_store_default_page_data:
__marginalia_lua_store_page_data:n
__marginalia_lua_check_page_data:n
__marginalia_lua_store_item_data:n
__marginalia_lua_check_item_data:n
__marginalia_lua_compute_items:
__marginalia_lua_write_problem_report:
__marginalia_lua_write_item_change_report:

The first 8 macros do not require expansion of parameters: they either have none, or process data not containing control sequences (read from the `.aux` file); hence `\lua_now:n` is used.

```
\cs_new:Npn\__marginalia_lua_store_default_page_data:
  {
    \lua_now:n{ marginalia.store_default_page_data() }
  }
\cs_new:Npn\__marginalia_lua_store_page_data:n #1
  {
    \lua_now:n{ marginalia.store_page_data('#1') }
  }
\cs_new:Npn\__marginalia_lua_check_page_data:n #1
  {
    \lua_now:n{ marginalia.check_page_data('#1') }
  }
\cs_new:Npn\__marginalia_lua_write_page_change_report:
  {
    \lua_now:n{ marginalia.write_page_change_report() }
  }
\cs_new:Npn\__marginalia_lua_store_item_data:n #1
  {
    \lua_now:n{ marginalia.store_item_data('#1') }
  }
\cs_new:Npn\__marginalia_lua_check_item_data:n #1
  {
    \lua_now:n{ marginalia.check_item_data('#1') }
  }
\cs_new:Npn\__marginalia_lua_compute_items:
  {
    \lua_now:n{ marginalia.compute_items() }
  }
\cs_new:Npn\__marginalia_lua_write_problem_report:
  {
    \lua_now:n{ marginalia.write_problem_report() }
  }
\cs_new:Npn\__marginalia_lua_write_item_change_report:
  {
    \lua_now:n{ marginalia.write_item_change_report() }
  }
```

(*End of definition for `\__marginalia_lua_store_default_page_data:` and others.*)

__marginalia_lua_load_item_data:n

The last macro will receive a control sequence parameter and so requires expansion; hence `\lua_now:e` is used.

```
\cs_new:Npn\__marginalia_lua_load_item_data:n #1
  {
    \lua_now:e{ marginalia.load_item_data('#1') }
  }
```

(*End of definition for `\__marginalia_lua_load_item_data:n`.*)

11.5 Processing data from the .aux file

\marginalia@pagedata

This command is used to store page data in the `.aux` file.

```
\NewDocumentCommand{\marginalia@pagedata}{ m }{
  \__marginalia_process_page_data:n{#1}
}
```

Initially __marginalia_process_page_data:n is set to __marginalia_lua_store_-page_data:n. Thus, when the .aux file is read, \marginalia@pagedata will pass the page data to the Lua backend to be stored.

```
\cs_set_eq:NN
  \__marginalia_process_page_data:n
  \__marginalia_lua_store_page_data:n
```

(*End of definition for* \marginalia@pagedata.)

\marginalia@itemdata This command is used to store data for each marginal content item in the .aux file.

```
\DeclareDocumentCommand{\marginalia@itemdata}{ m }{
  \__marginalia_process_item_data:n{#1}
}
```

(*End of definition for* \marginalia@itemdata.)

Initially __marginalia_process_item_data:n is set to __marginalia_lua_-store_item_data:n. Thus, when the .aux file is read, \marginalia@itemdata will pass the item data to the Lua backend to be stored.

```
\cs_set_eq:NN
  \__marginalia_process_item_data:n
  \__marginalia_lua_store_item_data:n
```

At the begindocument hook, the .aux file has been read and closed. The Lua backend now stores the geometry and computes the vertical shift for each item. Then the handle for the main .aux file is stored for use in this package.

```
\AddToHook{begindocument}{
  \__marginalia_lua_store_default_page_data:
  \__marginalia_lua_compute_items:
  \cs_set_eq:NN\l__marginalia_aux_iow\@mainaux
}
```

The enddocument/afterlastpage hook is before the .aux file is read back, so this is where __marginalia_process_page_data:n and __marginalia_process_-item_data:n are set, respectively, to __marginalia_lua_check_page_data:n and __marginalia_lua_check_item_data:n. Thus, when the .aux file is read back, \marginalia@pagedata and \marginalia@itemdata will pass data to the Lua backend to be checked for changes.

```
\AddToHook{enddocument/afterlastpage}{
  \cs_set_eq:NN
    \__marginalia_process_page_data:n
    \__marginalia_lua_check_page_data:n
  \cs_set_eq:NN
    \__marginalia_process_item_data:n
    \__marginalia_lua_check_item_data:n
  }
```

__marginalia_write_reports: All the reports of changes and/or problems are assembled in the Lua backend. This macro will write the reports as package warnings, using the following three messages, to which the Lua-assembled reports are passed as parameters:

```
\msg_new:nnn{marginalia}{placement_problem}
```

```
  { Problems~in~placement.~#1 }
\msg_new:nnn{marginalia}{item_change}
  { Changes~in~item~data.~#1 }
\msg_new:nnn{marginalia}{page_change}
  { Changes~in~page~data.~#1 }
\cs_new:Npn\__marginalia_write_reports:
  {
    \group_begin:
    \tl_set:Ne\l_tmpa_tl{\__marginalia_lua_write_problem_report:}
    \tl_if_blank:VF\l_tmpa_tl
      {
        \msg_warning:nne{marginalia}{placement_problem}{\tl_use:N\l_tmpa_tl}
      }
    \tl_set:Ne\l_tmpa_tl{\__marginalia_lua_write_item_change_report:}
    \tl_if_blank:VF\l_tmpa_tl
      {
        \msg_warning:nne{marginalia}{item_change}{\tl_use:N\l_tmpa_tl}
      }
    \tl_set:Ne\l_tmpa_tl{\__marginalia_lua_write_page_change_report:}
    \tl_if_blank:VF\l_tmpa_tl
      {
        \msg_warning:nne{marginalia}{page_change}{\tl_use:N\l_tmpa_tl}
      }
    \group_end:
  }
```

(*End of definition for* `\__marginalia_write_reports:`*.*)

Use the `enddocument/info` hook to write the reports of changes and/or problems.

```
\AddToHook{enddocument/info}{
  \__marginalia_write_reports:
}
```

11.6 Writing page data to the .aux file

To compute the positions of marginal content items, certain page layout data is required. And since all the computation takes place at the beginning of the document, it is necessary to write this information to the `.aux` file.

\g__marginalia_pagedatano_int

Global integer variable to index page data items written to the `.aux` file.

```
\int_new:N\g__marginalia_pagedatano_int
```

(*End of definition for* `\g__marginalia_pagedatano_int`*.*)

__marginalia_write_page_data

This command will be used to write the current page data to the `.aux` file. It is initially defined to do nothing, so that the use of `\marginalianewgeometry` in the preamble does not cause errors (because the `.aux` file is not available for writing until `begindocument/end`).

```
\cs_set_eq:NN\__marginalia_write_page_data:\prg_do_nothing:
\cs_new:Npn\__marginalia_write_page_data_real:
  {
    \int_gincr:N\g__marginalia_pagedatano_int
    \iow_now:Ne\l__marginalia_aux_iow{
      \token_to_str:N\marginalia@pagedata{
        pagedatano=\int_value:w\g__marginalia_pagedatano_int,
        abspageno=\int_eval:n{\g_shipout_readonly_int+1},
        hoffset=\int_value:w\hoffset,
        voffset=\int_value:w\voffset,
        paperheight=\int_value:w\paperheight,
        oddsidemargin=\int_value:w\oddsidemargin,
        evensidemargin=\int_value:w\evensidemargin,
        textwidth=\int_value:w\textwidth,
        columncount=\int_value:w\col@number,
        columnwidth=\int_value:w\columnwidth,
        columnsep=\int_value:w\columnsep,
        twoside=\bool_to_str:n{\legacy_if_p:n{@twoside}},
      }
    }
  }
```

At the begindocument/end hook, the .aux file has been opened for writing, and so the macro __marginalia_write_page_data: is enabled and the initial page data is written out.

```
\AddToHook{begindocument/end}
  {
    \cs_set_eq:NN
      \__marginalia_write_page_data:
      \__marginalia_write_page_data_real:
    \__marginalia_write_page_data:
  }
```

(*End of definition for* __marginalia_write_page_data.)

11.7 Marginal content item processing

11.7.1 Variables

Variables set by LaTeX.

\g__marginalia_itemno_int Global integer variable to index marginal content items.

```
\int_new:N\g__marginalia_itemno_int
```

(*End of definition for* \g__marginalia_itemno_int.)

\l__marginalia_item_box Box variable to hold the typeset marginal content item.

```
\box_new:N\l__marginalia_item_box
```

(*End of definition for* \l__marginalia_item_box.)

\l__marginalia_item_height_dim
\l__marginalia_item_depth_dim
Dimension variables to hold the height and depth of the typeset margin content item.

```
\dim_new:N\l__marginalia_item_height_dim
\dim_new:N\l__marginalia_item_depth_dim
```

(*End of definition for* \l__marginalia_item_height_dim *and* \l__marginalia_item_depth_dim.)

Variables set by Lua. The following variables will be set by the Lua backend via `tex.count` and `tex.dimen` when `\__marginalia_lua_load_item_data:n` is called.

`\l__marginalia_page_int`

Integer variable for the page on which the marginal content item appears. This variable will be made available via `\marginaliapage` within the ⟨*content*⟩ of `\marginalia`.

```
\int_new:N\l__marginalia_page_int
```

(*End of definition for* `\l__marginalia_page_int`.)

`\l__marginalia_column_computed_int`

Integer variable for the column next to which the marginal content item appears. This variable will be will be made available via `\marginaliacolumn` within the ⟨*content*⟩ of `\marginalia`.

```
\int_new:N\l__marginalia_column_computed_int
```

(*End of definition for* `\l__marginalia_column_computed_int`.)

`\l__marginalia_xshift_computed_dim`
`\l__marginalia_yshift_computed_dim`

Dimension variables to hold the differences in x and y coordinates between the call to `\marginalia` and the position where the marginal content item should appear.

```
\dim_new:N\l__marginalia_xshift_computed_dim
\dim_new:N\l__marginalia_yshift_computed_dim
```

(*End of definition for* `\l__marginalia_xshift_computed_dim` *and* `\l__marginalia_yshift_computed_-dim`.)

`\l__marginalia_side_computed_int`

Integer variable to indicate the side of the text block or column on which the marginal content item should be placed: 0 = right and 1 = left.

```
\int_new:N\l__marginalia_side_computed_int
```

(This variable could be a boolean, but an integer is used because there is no canonical access to booleans from Lua.)

(*End of definition for* `\l__marginalia_side_computed_int`.)

`\l__marginalia_marginno_computed_int`

Integer variable to indicate in which margin the content will be be placed, to enable quick selection of width and style: 0 = recto outer, 1 = recto inner, 2 = verso outer, 3 = verso inner, 4 = right between, 5 = left between.

```
\int_new:N\l__marginalia_marginno_computed_int
```

(*End of definition for* `\l__marginalia_marginno_computed_int`.)

`\l__marginalia_enabled_computed_int`

Integer variable to indicate whether the marginal content item is enabled: 0 = disabled, 1 = enabled.

```
\int_new:N\l__marginalia_enabled_computed_int
```

(This variable could be a boolean, but an integer is used because there is no canonical access to booleans from Lua.)

(*End of definition for* `\l__marginalia_enabled_computed_int`.)

11.7.2 Core macro

__marginalia_process_item:nn

This macro does most of the work in setting the marginal content item. The first parameter is ⟨*options*⟩, the second is ⟨*content*⟩.

```
\cs_new:Npn\__marginalia_process_item:nn #1#2
  {
```

First, increment the index, then enter a group where all the action will happen.

```
      \int_gincr:N\g__marginalia_itemno_int
      \group_begin:
```

Process ⟨*options*⟩. These settings apply locally inside the group.

```
        \keys_set:nn{marginalia}{ #1 }
```

Get item data from the Lua backend: the integer variables `\l__marginalia_page_int`, `\l__marginalia_column_computed_int`, `\l__marginalia_side_computed_int`, `\l__marginalia_enabled_computed_int`, and the dimension variables `\l__marginalia_xshift_computed_dim`, and `\l__marginalia_yshift_computed_dim` are set by Lua via `tex.count` and `tex.dimen`. If no data is available (if, for instance, no data has been stored from a previous run), default values will be set by Lua. On later runs, the Lua backend will supply the values computed from the data written to the `.aux` file on the previous run.

```
        \__marginalia_lua_load_item_data:n
          { \int_value:w\g__marginalia_itemno_int }
```

Choose the correct auxiliary function for typesetting, depending on which mode TEX is in.

```
        \mode_if_math:TF
          {
            \cs_set_eq:NN
              \__marginalia_typeset:n
              \__marginalia_typeset_mmode:n
          }
          {
            \legacy_if:nT{@inlabel}
              { \leavevmode }
            \mode_if_horizontal:TF
              {
                \cs_set_eq:NN
                  \__marginalia_typeset:n
                  \__marginalia_typeset_hmode:n
              }
              {
                \cs_set_eq:NN
                  \__marginalia_typeset:n
                  \__marginalia_typeset_vmode:n
              }
          }
```

Choose the correct box in which to typeset the item. `\l__marginalia_valign_int` can only be 1 or 2, so take 2 to signify bottom-aligned, anything else signifies top-aligned.

```
        \int_compare:nNnTF{\l__marginalia_valign_int}={2}
          {
            \cs_set_eq:NN\__marginalia_item_box_set:Nn\vbox_set:Nn
          }
          {
            \cs_set_eq:NN\__marginalia_item_box_set:Nn\vbox_set_top:Nn
          }
```

Choose the correct horizontal separation, width, and style for the item.

```
      \__marginalia_set_xsep_width_style:
```

Typeset the ⟨*content*⟩ into `\l__marginalia_item_box`. Use `\@parboxrestore` for brevity, even though `\hsize` and `\linewidth` are subsequently set to `\l__marginalia_-width_dim`. Make available `\marginaliapage` and `\marginaliacolumn`.

```
      \__marginalia_tagging_socket:n {marginpar/begin}
      \__marginalia_item_box_set:Nn\l__marginalia_item_box{
        \@parboxrestore
        \__marginalia_tagging_socket:n {para/restore}
        \normalfont\normalsize

        \tl_use:N\l__marginalia_style_tl
        \dim_set_eq:NN\hsize\l__marginalia_width_dim
        \dim_set_eq:NN\linewidth\hsize

        \cs_set_eq:NN\marginaliapage\l__marginalia_page_int
        \cs_set_eq:NN\marginaliacolumn\l__marginalia_column_computed_int

        \group_begin:
        \ignorespaces
        #2
        \par
        \group_end:
      }
      \__marginalia_tagging_socket:n{marginpar/end}
```

Measure `\l__marginalia_item_box`.

```
      \dim_set:Nn\l__marginalia_item_height_dim
        {\box_ht:N\l__marginalia_item_box}
      \dim_set:Nn\l__marginalia_item_depth_dim
        {\box_dp:N\l__marginalia_item_box}
```

Everything is now ready to place the item on the page and write the necessary data to the `.aux` file. Use the chosen auxiliary function for typesetting, and immediately use `\savepos` to store the callout position.

```
      \__marginalia_typeset:n{
        \savepos
```

Write the item data to the `.aux` file. All tokens that will change for future items, and which are currently meaningful, are expanded now; the remainder will be expanded at shipout time, when *they* are meaningful.

```
        \iow_shipout_e:Ne\l__marginalia_aux_iow{
          \token_to_str:N\marginalia@itemdata{
            itemno=\int_value:w\g__marginalia_itemno_int,
            abspageno=\exp_not:N\int_eval:n{\g_shipout_readonly_int},
            pageno=\exp_not:N\int_value:w\c@page,
            type=\str_use:N\int_value:w\l__marginalia_type_int,
            xpos=\exp_not:N\int_value:w\lastxpos,
            ypos=\exp_not:N\int_value:w\lastypos,
            height=\int_value:w\l__marginalia_item_height_dim,
            depth=\int_value:w\l__marginalia_item_depth_dim,
            pos=\int_value:w\l__marginalia_pos_int,
            column=\int_value:w\l__marginalia_column_int,
            yshift=\int_value:w\l__marginalia_default_yshift_dim,
            ysep~above=\int_value:w\l__marginalia_ysep_above_dim,
            ysep~below=\int_value:w\l__marginalia_ysep_below_dim,
            ysep~page~top=\int_value:w\l__marginalia_ysep_page_top_dim,
            ysep~page~bottom=\int_value:w\l__marginalia_ysep_page_bottom_dim,
          }
        }
```

Finally, if the item is enabled, typeset it onto the page: shift the item by

$$\left|\texttt{\textbackslash l\_\_marginalia\_xshift\_computed\_dim}\right| + \left|\texttt{\textbackslash l\_\_marginalia\_xsep\_dim}\right|$$

to the right in an `\rlap` or to the left in an `\llap`, depending on `\l__marginalia_side_computed_int`, then use `\__marginalia_place_item_box` for the vertical placement.

```
      \int_if_zero:nF{\l__marginalia_enabled_computed_int}
        {
          \int_if_zero:nTF{\l__marginalia_side_computed_int}
            {
              \rlap{
                \kern\l__marginalia_xshift_computed_dim
                \kern\l__marginalia_xsep_dim
                \__marginalia_place_item_box:
              }
            }
            {
              \llap{
                \__marginalia_place_item_box:
                \kern\l__marginalia_xsep_dim
                \kern-\l__marginalia_xshift_computed_dim
              }
            }
        }
      }
```

Close the group started near the beginning of `\__marginalia_process_item:nn`.

```
    \group_end:
  }
```

(*End of definition for* `\__marginalia_process_item:nn`.)

11.7.3 Width and style selection

`\__marginalia_set_xsep_width_style` Set `\l__marginalia_xsep_dim`, `\l__marginalia_width_dim`, and `\l__marginalia_style_tl`, based on `\l__marginalia_marginno_computed_int`.

```
\cs_new:Npn\__marginalia_set_xsep_width_style:
  {
    \int_case:nn{\l__marginalia_marginno_computed_int}
      {
        {0}
        {
          \cs_set_eq:NN\l__marginalia_xsep_dim
            \l__marginalia_xsep_recto_outer_dim
          \cs_set_eq:NN\l__marginalia_width_dim
            \l__marginalia_width_recto_outer_dim
          \cs_set_eq:NN\l__marginalia_style_tl
            \l__marginalia_style_recto_outer_tl
        }
        {1}
        {
          \cs_set_eq:NN\l__marginalia_xsep_dim
            \l__marginalia_xsep_recto_inner_dim
          \cs_set_eq:NN\l__marginalia_width_dim
            \l__marginalia_width_recto_inner_dim
          \cs_set_eq:NN\l__marginalia_style_tl
            \l__marginalia_style_recto_inner_tl
        }
        {2}
        {
          \cs_set_eq:NN\l__marginalia_xsep_dim
            \l__marginalia_xsep_verso_outer_dim
          \cs_set_eq:NN\l__marginalia_width_dim
            \l__marginalia_width_verso_outer_dim
          \cs_set_eq:NN\l__marginalia_style_tl
            \l__marginalia_style_verso_outer_tl
        }
        {3}
        {
          \cs_set_eq:NN\l__marginalia_xsep_dim
            \l__marginalia_xsep_verso_inner_dim
          \cs_set_eq:NN\l__marginalia_width_dim
            \l__marginalia_width_verso_inner_dim
          \cs_set_eq:NN\l__marginalia_style_tl
            \l__marginalia_style_verso_inner_tl
        }
        {4}
        {
          \cs_set_eq:NN\l__marginalia_xsep_dim
            \l__marginalia_xsep_right_between_dim
          \cs_set_eq:NN\l__marginalia_width_dim
            \l__marginalia_width_right_between_dim
          \cs_set_eq:NN\l__marginalia_style_tl
            \l__marginalia_style_right_between_tl
        }
        {5}
        {
          \cs_set_eq:NN\l__marginalia_xsep_dim
            \l__marginalia_xsep_left_between_dim
          \cs_set_eq:NN\l__marginalia_width_dim
            \l__marginalia_width_left_between_dim
          \cs_set_eq:NN\l__marginalia_style_tl
            \l__marginalia_style_left_between_tl
        }
      }
  }
```

(*End of definition for* `\__marginalia_set_xsep_width_style`*.*)

11.7.4 Auxiliary placement macros

__marginalia_place_item_box: Place the item that has been set in `\l__marginalia_item_box`, vertically shifted by `\l__marginalia_yshift_computed_dim` and `\smash`ed to avoid altering vertical spacing in the main text.

```
\cs_new:Npn\__marginalia_place_item_box:
  {
    \smash
      {
        \box_move_up:nn{\l__marginalia_yshift_computed_dim}
          {
            \box_use:N\l__marginalia_item_box
          }
      }
  }
```

(*End of definition for* `\__marginalia_place_item_box:`.)

__marginalia_typeset_mmode:n
__marginalia_typeset_hmmode:n
__marginalia_typeset_vmode:n

These three macros handle typsetting in math mode, horizontal mode, and vertical mode. Nothing special needs to be done in math mode. In horizontal mode, `\@bsphack`…`\@bsphack` avoids double spacing. In vertical mode, `\if@nobreak` is saved, a new paragraph is started, the item is typeset, the paragraph is ended, a vertical skip of −`\baselineskip` is added, which should 'hide' that invisible paragraph, and `\if@nobreak` is restored to the saved value.

```
\cs_new:Npn\__marginalia_typeset_mmode:n #1
  {
    #1
  }
\cs_new:Npn\__marginalia_typeset_hmode:n #1
  {
    \@bsphack
    #1
    \@esphack
  }
\cs_new:Npn\__marginalia_typeset_vmode:n #1
  {
    \bool_set:Nn\l_tmpa_bool{ \legacy_if_p:n{@nobreak} }
    \nobreak\noindent #1\par
    \skip_vertical:n{-\baselineskip}
    \legacy_if_gset:nn{ @nobreak }{ \l_tmpa_bool }
  }
```

(*End of definition for* `\__marginalia_typeset_mmode:n`*,* `\__marginalia_typeset_hmmode:n`*, and* `\__marginalia_typeset_vmode:n`*.*)

11.8 User commands

Finally, set up the commands for the user.

\marginalia This is the main user command for creating a marginal content item. This macro does nothing but hand off to `\__marginalia_process_item:nn`.

```
\NewDocumentCommand{\marginalia}{ O{} +m }
  {
    \__marginalia_process_item:nn{#1}{#2}
  }
```

(*End of definition for* `\marginalia`*. This function is documented on page 4.*)

`\marginaliasetup` The user command to set the configuration.

```
\NewDocumentCommand{\marginaliasetup}{ m }
{
  \keys_set:nn{marginalia}{ #1 }
}
```

(*End of definition for* `\marginaliasetup`*. This function is documented on page 4.*)

`\marginalianewgeometry` The user command to signal that the page geometry has been changed.

```
\NewDocumentCommand{\marginalianewgeometry}{}
{
  \__marginalia_write_page_data:
}
```

(*End of definition for* `\marginalianewgeometry`*. This function is documented on page 4.*)

```
⟨/package⟩
```

12 Implementation (Lua backend)

```
⟨*lua⟩
```

12.1 Global variables

Global tables for page_data and item_data.

```
local PAGE_DATA_MAIN_TABLE = {}
local ITEM_DATA_MAIN_TABLE = {}
```

Global tables for compiling reports.

```
local PROBLEM_REPORT_TABLE = {}
local PAGE_CHANGE_REPORT_TABLE = {}
local ITEM_CHANGE_REPORT_TABLE = {}
```

Global configuration for reports.

```
local PROBLEM_REPORT_MAX_LENGTH = 40
local PAGE_CHANGE_REPORT_MAX_LENGTH = 10
local ITEM_CHANGE_REPORT_MAX_LENGTH = 10
```

12.2 Constants

Type constants. These match the possible values for the type key.

```
local TYPE_NORMAL = 1
local TYPE_FIXED = 2
local TYPE_OPTFIXED = 3
```

Position constants. These match the possible values for the pos key.

```
local POS_AUTO = 1
local POS_REVERSE = 2
local POS_LEFT = 3
local POS_RIGHT = 4
local POS_NEAREST = 5
```

12.3 Keys for tables

The strings listed in this subsection are constants used to index the tables. Also listed are the types of values that are indexed by each key. Note that values listed below as `dimensions` are actually integers, giving the dimension in TeX scaled points (sp)

12.3.1 Keys for both page and item data tables

Integer: Absolute page number in output file (not on-page number), used in both page_-data and item_data tables

```
local KEY_ABSPAGENO = 'abspageno'
```

Boolean: Used to mark page_data or item_data as checked when the .aux file is read back at the end of the document

```
local KEY_CHECKED = 'checked'
```

12.3.2 Keys for page data tables, layout etc.

Integer: Used only to distinguish instances of data written to .aux file

```
local KEY_PAGEDATANO = 'pagedatano'
```

Dimensions: Value of next two will always be equivalent of 1 in, but it is simpler to keep all geometry data together.

```
local KEY_HOFFSETORIGIN = 'hoffsetorigin'
local KEY_VOFFSETORIGIN = 'voffsetorigin'
```

Dimensions: corresponding to obvious LaTeX dimensions

```
local KEY_HOFFSET = 'hoffset'
local KEY_VOFFSET = 'voffset'
local KEY_PAPERHEIGHT = 'paperheight'
local KEY_ODDSIDEMARGIN = 'oddsidemargin'
local KEY_EVENSIDEMARGIN = 'evensidemargin'
local KEY_TEXTWIDTH = 'textwidth'
local KEY_COLUMNWIDTH = 'columnwidth'
local KEY_COLUMNSEP = 'columnsep'
```

Integer: either 1 or 2, depending on whether LaTeX was in one- or two-column mode

```
local KEY_COLUMNCOUNT = 'columncount'
```

Boolean: true iff LaTeX is in twoside mode

```
local KEY_TWOSIDE = 'twoside'
```

12.3.3 Keys for item data tables

Integer: Used to identify data with item

```
local KEY_ITEMNO = 'itemno'
```

Integer: On-page number

```
local KEY_PAGENO = 'pageno'
```

Dimensions: x and y positions of call to `\marginalia`

```
local KEY_XPOS = 'xpos'
local KEY_YPOS = 'ypos'
```

Dimensions: Height and depth of typeset item

```
local KEY_HEIGHT = 'height'
local KEY_DEPTH = 'depth'
```

Integer: Specified type, following `TYPE_*`

```
local KEY_TYPE = 'type'
```

Integer: corresponds to value of `pos` key: $0 =$ `auto`, $1 =$ `reverse`, $2 =$ `left`, $3 =$ `right`, $4 =$ `nearest`

```
local KEY_POS = 'pos'
```

Integer: corresponds to value of `column` key: $-1 =$ `auto`, $0 =$ `one`, $1 =$ `left`, $2 =$ `right`

```
local KEY_COLUMN = 'column'
```

Dimension: specified vertical shift

```
local KEY_YSHIFT = 'yshift'
```

Dimensions: specified vertical separations

```
local KEY_YSEP_ABOVE = 'ysep above'
local KEY_YSEP_BELOW = 'ysep below'
local KEY_YSEP_PAGE_TOP = 'ysep page top'
local KEY_YSEP_PAGE_BOTTOM = 'ysep page bottom'
```

The preceding keys refer to values that will be supplied from LaTeX. The remaining values will be computed in Lua and passed back to LaTeX.

Integer: column in which the call to `\marginalia` was located: $0 =$ one-column, $1 =$ left, $2 =$ right

```
local KEY_COLNO_COMPUTED = 'colno computed'
```

Dimension: Horizontal shift between the call to `\marginalia` and the margin in which the item should be located

```
local KEY_XSHIFT_COMPUTED = 'xshift computed'
```

Dimension: Computed vertical shift

```
local KEY_YSHIFT_COMPUTED = 'yshift computed'
```

Integer: Side of text on which the item will appear: $0 =$ right, $1 =$ left

```
local KEY_SIDE_COMPUTED = 'side computed'
```

Integer: Number of margin in which the item will appear, $0 =$ recto outer, $1 =$ recto inner, $2 =$ verso outer, $3 =$ verso inner, $4 =$ right between, $5 =$ left between

```
local KEY_MARGINNO_COMPUTED = 'marginno computed'
```

Boolean: Whether the item will actually appear on the page

```
local KEY_ENABLED_COMPUTED = 'enabled computed'
```

12.4 Utility functions

list_filter

Take a list `t` and remove from it any elements for which the function `f` does not return true. (The index `j` is always the destination index to which a ‘keep’ element is moved.)[7]

[7] Code adapted from https://stackoverflow.com/a/53038524/8990243.

```
local function list_filter(t, f)
  local j = 1
  local n = #t

  for i=1,n do
    if (f(t[i])) then
      if (i ~= j) then
        t[j] = t[i]
        t[i] = nil
      end
      j = j + 1
    else
      t[i] = nil
    end
  end

end
```

(*End of definition for* `list_filter`.)

list_filter

Return boolean true iff `s` is exactly the string ‘`true`’.

```
local function toboolean(s)
  return s == "true"
end
```

(*End of definition for* `list_filter`.)

get_data_page_number

Take a item or page data and return a human-readable string indicating the page to which the data pertains.

```
local function get_data_page_number(data)
  local pageno = data[KEY_PAGENO]
  if pageno ~= nil then
    return 'p' .. pageno .. ' (' .. data[KEY_ABSPAGENO] .. ')'
  else
    return data[KEY_ABSPAGENO]
  end
end
```

(*End of definition for* `get_data_page_number`.)

12.5 Generic page/item data functions

parse_data

Parse `keyvalue_string` and return the corresponding data as a table. The `keyvalue_string` is expected to be of precisely the kind written to the `.aux` file as the parameter of `\marginalia@pagedata` or `\marginalia@notedata`.

Ignore any keys in `keyvalue_string` that are not listed in `conversion_table`. Fill in any missing value with values from `defaults_table`.

`conversion_table` is indexed by possible keys, with values equal to functions to convert the corresponding value string to the value that should appear in the returned table.

defaults_table is indexed by keys that *will* appear in the returned table, using the corresponding value unless it was given in keyvalue_string and the key appeared in conversion_table.

```
local function parse_data(keyvalue_string,conversion_table,defaults_table)

  local key
  local value
  local result = {}

  for s in string.gmatch(keyvalue_string,'([^,]+)') do

    key,value = string.match(s,'^(.+)=(.+)$')
    local conv = conversion_table[key]
    if conv ~= nil then
      result[key] = conv(value)
    end

  end

  for key,value in pairs(defaults_table) do
    if not(result[key] ~= nil) then
      result[key] = value
    end
  end

  return result

end
```

(*End of definition for* parse_data.)

check_data Check keyvalue_string against stored data. If it is new or has changed, append a report to report_table. Set the KEY_CHECKED of the data item to true.

The keyvalue_string is processed using conversion_table and defaults_table as per the parse_data function. The resulting table is compared to the table in data_table with the same value whose key is data_table_key. The tables are compared using the fields indexed by keys in conversion_table.

```
local function check_data(keyvalue_string,conversion_table,defaults_table,
                          data_table,data_table_key_field,report_table)

  local new_data = parse_data(keyvalue_string,
                              conversion_table,defaults_table)

  local data_table_key = new_data[data_table_key_field]

  local stored_data = data_table[data_table_key]
  if stored_data == nil then
    table.insert(
      report_table,
      get_data_page_number(new_data) .. ' New'
    )
  else
    local change_report = ''
    for k,_ in pairs(conversion_table) do
      if stored_data[k] ~= new_data[k] then
        change_report = change_report
          .. ' ' .. k .. ':' ..
          tostring(stored_data[k]) .. '->' .. tostring(new_data[k])
      end
    end
    if change_report ~= '' then
      table.insert(
        report_table,
        get_data_page_number(new_data) .. ' ' .. change_report
      )
    end
    stored_data[KEY_CHECKED] = true
  end

end
```

(*End of definition for* `check_data`.)

`check_removed_data` Check whether data have been removed from `data_table`, which corresponds to some entry having the value of `KEY_CHECKED` being false. In this case, append a report to `report_table`.

```
local function check_removed_data(data_table,report_table)
  for _,data in pairs(data_table) do
    if not data[KEY_CHECKED] then
      table.insert(
        report_table,
        ' Removed'
      )
      break
    end
  end
end
```

(*End of definition for* `check_removed_data`.)

12.6 Processing of page data from .aux file

Conversion and default tables.

```
local PAGE_DATA_CONVERSION_TABLE = {
  [KEY_PAGEDATANO] = tonumber,
  [KEY_ABSPAGENO] = tonumber,
  [KEY_HOFFSETORIGIN] = tonumber,
  [KEY_VOFFSETORIGIN] = tonumber,
  [KEY_HOFFSET] = tonumber,
  [KEY_VOFFSET] = tonumber,
  [KEY_PAPERHEIGHT] = tonumber,
  [KEY_ODDSIDEMARGIN] = tonumber,
  [KEY_EVENSIDEMARGIN] = tonumber,
  [KEY_COLUMNCOUNT] = tonumber,
  [KEY_COLUMNWIDTH] = tonumber,
  [KEY_COLUMNSEP] = tonumber,
  [KEY_TEXTWIDTH] = tonumber,
  [KEY_TWOSIDE] = toboolean,
}
local PAGE_DATA_DEFAULT_TABLE = {
  [KEY_PAGEDATANO] = 0,
  [KEY_ABSPAGENO] = 0,
  [KEY_HOFFSETORIGIN] = tex.sp('1in'),
  [KEY_VOFFSETORIGIN] = tex.sp('1in'),
  [KEY_HOFFSET] = tex.dimen['hoffset'],
  [KEY_VOFFSET] = tex.dimen['voffset'],
  [KEY_PAPERHEIGHT] = tex.dimen['paperheight'],
  [KEY_ODDSIDEMARGIN] = tex.dimen['oddsidemargin'],
  [KEY_EVENSIDEMARGIN] = tex.dimen['evensidemargin'],
  [KEY_TEXTWIDTH] = tex.dimen['textwidth'],
  [KEY_COLUMNWIDTH] = tex.dimen['columnwidth'],
  [KEY_COLUMNSEP] = tex.dimen['columnsep'],
  [KEY_COLUMNCOUNT] = 1,
  [KEY_TWOSIDE] = false,
  [KEY_CHECKED] = false,
}
```

store_page_data Store page data supplied by `keyvalue_string` in `PAGE_DATA_MAIN_TABLE`.

```
local function store_page_data(keyvalue_string)

  local page_data = parse_data(keyvalue_string,
                               PAGE_DATA_CONVERSION_TABLE,
                               PAGE_DATA_DEFAULT_TABLE)

  PAGE_DATA_MAIN_TABLE[page_data[KEY_PAGEDATANO]] = page_data

end
```

(*End of definition for* `store_page_data`.)

store_default_page_data Store default page data in `PAGE_DATA_MAIN_TABLE`, so that there is some data to work with when computing item positions, even on a first run, when no page data has been written to the `.aux` file.

```
local function store_default_page_data()

  default_page_data = parse_data('',
                                 PAGE_DATA_CONVERSION_TABLE,
                                 PAGE_DATA_DEFAULT_TABLE)

  default_page_data[KEY_ABSPAGENO] = 1
  default_page_data[KEY_CHECKED] = true

  PAGE_DATA_MAIN_TABLE[0] = default_page_data

end
```

(*End of definition for* `store_default_page_data`.)

check_page_data Check whether page_data supplied by keyvalue_string differs from that in `PAGE_DATA_MAIN_TABLE`, appending reports to `PAGE_CHANGE_REPORT_TABLE` if so.

```
local function check_page_data(keyvalue_string)
```

```

  check_data(keyvalue_string,
             PAGE_DATA_CONVERSION_TABLE,PAGE_DATA_DEFAULT_TABLE,
             PAGE_DATA_MAIN_TABLE,KEY_PAGEDATANO,
             PAGE_CHANGE_REPORT_TABLE)

end
```

(*End of definition for* `check_page_data`.)

12.7 Processing of item data from .aux file

Conversion and default tables.

```
local ITEM_DATA_CONVERSIONS = {
  [KEY_ITEMNO] = tonumber,
  [KEY_ABSPAGENO] = tonumber,
  [KEY_PAGENO] = tonumber,
  [KEY_XPOS] = tonumber,
  [KEY_YPOS] = tonumber,
  [KEY_HEIGHT] = tonumber,
  [KEY_DEPTH] = tonumber,
  [KEY_TYPE] = tonumber,
  [KEY_POS] = tonumber,
  [KEY_COLUMN] = tonumber,
  [KEY_YSHIFT] = tonumber,
  [KEY_YSEP_ABOVE] = tonumber,
  [KEY_YSEP_BELOW] = tonumber,
  [KEY_YSEP_PAGE_TOP] = tonumber,
  [KEY_YSEP_PAGE_BOTTOM] = tonumber,
  [KEY_CHECKED] = toboolean,
}
local ITEM_DATA_DEFAULTS = {
  [KEY_ITEMNO] = 0,
  [KEY_ABSPAGENO] = 1,
  [KEY_PAGENO] = 1,
  [KEY_XPOS] = 0,
  [KEY_YPOS] = 0,
  [KEY_HEIGHT] = 0,
  [KEY_DEPTH] = 0,
  [KEY_TYPE] = 0,
  [KEY_POS] = 0,
  [KEY_COLUMN] = -1,
  [KEY_YSHIFT] = 0,
  [KEY_YSEP_ABOVE] = tex.dimen['marginparpush'],
  [KEY_YSEP_BELOW] = tex.dimen['marginparpush'],
  [KEY_YSEP_PAGE_TOP] = tex.dimen['marginparpush'],
  [KEY_YSEP_PAGE_BOTTOM] = tex.dimen['marginparpush'],
  [KEY_COLNO_COMPUTED] = 0,
  [KEY_XSHIFT_COMPUTED] = 0,
  [KEY_YSHIFT_COMPUTED] = 0,
  [KEY_SIDE_COMPUTED] = 0,
  [KEY_MARGINNO_COMPUTED] = 0,
  [KEY_ENABLED_COMPUTED] = true,
  [KEY_CHECKED] = false,
}
```

ITEM_DATA_DEFAULTS is also used by load_item_data when no stored item data is found in ITEM_DATA_MAIN_TABLE.

store_item_data Store item_data supplied by keyvalue_string in ITEM_DATA_MAIN_TABLE.

```
local function store_item_data(keyvalue_string)

  local item = parse_data(keyvalue_string,
                          ITEM_DATA_CONVERSIONS,
                          ITEM_DATA_DEFAULTS)

  ITEM_DATA_MAIN_TABLE[item[KEY_ITEMNO]] = item

end
```

(*End of definition for* store_item_data.)

check_item_data Check whether item_data supplied by keyvalue_string differs from that in ITEM_-DATA_MAIN_TABLE, appending reports to ITEM_CHANGE_REPORT_TABLE if so.

```
local function check_item_data(keyvalue_string)

  check_data(keyvalue_string,
             ITEM_DATA_CONVERSIONS,ITEM_DATA_DEFAULTS,
             ITEM_DATA_MAIN_TABLE,KEY_ITEMNO,
             ITEM_CHANGE_REPORT_TABLE)

end
```

(*End of definition for* check_item_data.)

12.8 Writing reports

write_report Write the data contained in report_table to TEX in a format suitable for a package warning. The written text will contain at most max_length items.

```
local function write_report(report_table,max_length)

  if #report_table > 0 then
    local report_text
    local report_length

    if #report_table <= max_length then
      report_length = #report_table
      report_text = ' Here they are:\n'
    else
      report_length = max_length
      report_text = ' Here are the first ' .. report_length .. ':\n'
    end

    for i=1,report_length do
      report_text = report_text .. report_table[i]
      if i < report_length then
        report_text = report_text .. '\n'
      end
    end

    tex.print(report_text)
  end

end
```

(*End of definition for* `write_report`*.*)

`write_problem_report` Write a report about placement problems to TeX in a format suitable for a package warning.

```
local function write_problem_report()

  write_report(PROBLEM_REPORT_TABLE,PROBLEM_REPORT_MAX_LENGTH)

end
```

(*End of definition for* `write_problem_report`*.*)

`write_item_change_report` Write a report about changes in item data to TeX in a format suitable for a package warning.

```
local function write_item_change_report()

  check_removed_data(ITEM_DATA_MAIN_TABLE,ITEM_CHANGE_REPORT_TABLE)
  write_report(ITEM_CHANGE_REPORT_TABLE,ITEM_CHANGE_REPORT_MAX_LENGTH)

end
```

(*End of definition for* `write_item_change_report`*.*)

`write_page_change_report` Write a report about changes in page data to TeX in a format suitable for a package warning.

```
local function write_page_change_report()

  check_removed_data(PAGE_DATA_MAIN_TABLE,PAGE_CHANGE_REPORT_TABLE)
  write_report(PAGE_CHANGE_REPORT_TABLE,PAGE_CHANGE_REPORT_MAX_LENGTH)

end
```

(*End of definition for* `write_page_change_report`*.*)

12.9 Computing horizontal positions

It is necessary to determine whether an item should be placed on the right or left of the text block, and in which column it lies. The following lookup tables are used.

The value found in `RIGHTSIDE_LOOKUP_TABLE` is either `true` (right) or `false` (left). It is indexed by whether the item is on a recto page (`true`/`false`), whether it pertains to single-column text, the left column, or the right colum (`0`/`1`/`2`), and the value of `pos` being either `auto` or `reverse`.

```
local RIGHTSIDE_LOOKUP_TABLE = {
  [true] = {
    [0] = {
      [POS_AUTO] = true,
      [POS_REVERSE] = false,
    },
    [1] = {
      [POS_AUTO] = false,
      [POS_REVERSE] = true,
    },
    [2] = {
      [POS_AUTO] = true,
      [POS_REVERSE] = false,
    },
  },
  [false] = {
    [0] = {
      [POS_AUTO] = false,
      [POS_REVERSE] = true,
    },
    [1] = {
      [POS_AUTO] = true,
      [POS_REVERSE] = false,
    },
    [2] = {
      [POS_AUTO] = false,
      [POS_REVERSE] = true,
    },
  },
}
```

The value found in `MARGINNO_LOOKUP_TABLE` ranges from `0` to `5` (see `KEY_MARGINNO_-COMPUTED` for the meaning of these values). It is indexed by whether the item is on a recto page (`true`/`false`), whether it pertains to single-column text, the left column, or the right colum (`0`/`1`/`2`), and whether it is to be placed on the right of the text block (`true`/`false`).

```
local MARGINNO_LOOKUP_TABLE = {
  [true] = {
    [0] = {
      [false] = 1,
      [true] = 0,
    },
    [1] = {
      [false] = 1,
      [true] = 5,
    },
    [2] = {
      [false] = 4,
      [true] = 0,
    },
  },
  [false] = {
    [0] = {
      [false] = 2,
      [true] = 3,
    },
    [1] = {
      [false] = 2,
      [true] = 5,
    },
    [2] = {
      [false] = 4,
      [true] = 3,
    },
  },
}
```

compute_items_horizontal For every `item_data` in `item_data_list`, compute the fields relevant to horizontal positioning, namely `KEY_COLNO_COMPUTED`, `KEY_XSHIFT_COMPUTED`, `KEY_SIDE_COMPUTED`, based on the layout information in page_data. Every item described in `item_data_list` is assumed to be on the same page.

```
local function compute_items_horizontal(item_data_list,page_data)
```

Immediately return if `item_data_list` is empty, to avoid edge cases.

```
  if #item_data_list == 0 then
    return
  end
```

Information used frequently and which is the same for every item.

```
  local pageno = item_data_list[1][KEY_PAGENO]
  local twoside = page_data[KEY_TWOSIDE]
  local recto = ((pageno % 2) == 1) or (not twoside)
  local columncount = page_data[KEY_COLUMNCOUNT]
```

Tables to contain the x-coordinates of left edge, right edge, and middle of the current text, whether a single column (index 0), the left column (index 1), or the right column (index 2).

```
  local x_textleft = {}
  local x_textright = {}
  local x_textmiddle = {}
```

First, compute necessary dimensions for single-column text, since most of these calculations would be used anyway for two-column text. The terms used in calculating `x_textleft[0]` respectively take one to the origin of `\hoffset`, to the origin of `\oddsidemargin` and `\evensidemargin`, and to the left-hand side of the text block.

```
  if recto then
    x_textleft[0] = (
      page_data[KEY_HOFFSETORIGIN]
      + page_data[KEY_HOFFSET]
      + page_data[KEY_ODDSIDEMARGIN]
    )
    x_textright[0] = (
      x_textleft[0]
      + page_data[KEY_TEXTWIDTH]
    )
  else
    x_textleft[0] = (
      page_data[KEY_HOFFSETORIGIN]
      + page_data[KEY_HOFFSET]
      + page_data[KEY_EVENSIDEMARGIN]
    )
    x_textright[0] = (
      x_textleft[0]
      + page_data[KEY_TEXTWIDTH]
    )
  end
  x_textmiddle[0] = (x_textleft[0] + x_textright[0])/2


  if columncount == 1 then
```

If the page is one-column, the field KEY_COLNO_COMPUTED can be set immediately for every item_data.

```
    for i=1,#item_data_list do
      item_data_list[i][KEY_COLNO_COMPUTED] = 0
    end
  else
```

If the page is two-column, calculate the x-coordinates of the left and right edges and the mid-point of each column.

```
    x_textleft[1] = x_textleft[0]
    x_textright[1] = (
      x_textleft[1]
      + page_data[KEY_COLUMNWIDTH]
    )
    x_textmiddle[1] = (x_textleft[1] + x_textright[1])/2

    x_textleft[2] = (
      x_textright[1]
      + page_data[KEY_COLUMNSEP]
    )
    x_textright[2] = (
      x_textleft[2]
      + page_data[KEY_COLUMNWIDTH]
    )
    x_textmiddle[2] = (x_textleft[2] + x_textright[2])/2

```

Calculate the cut-off (mid-way between the columns) that distinguishes items from left and right columns.

```
    local left_column_x_limit = (
      x_textright[1]
      + .5*page_data[KEY_COLUMNSEP]
    )
```

Now set the field KEY_COLNO_COMPUTED for each item.

```
    for i=1,#item_data_list do
      local item_data = item_data_list[i]

      if item_data[KEY_COLUMN] >= 0 then
        item_data[KEY_COLNO_COMPUTED] = item_data[KEY_COLUMN]
      else
        if item_data[KEY_XPOS] <= left_column_x_limit then
          item_data[KEY_COLNO_COMPUTED] = 1
        else
          item_data[KEY_COLNO_COMPUTED] = 2
        end
      end
    end
```

```
  end
```

For every item_data in item_data_list, compute and set the fields `KEY_SIDE_COMPUTED`, `KEY_XSHIFT_COMPUTED`, and `KEY_MARGINNO_COMPUTED`.

```
  for i=1,#item_data_list do
    local item = item_data_list[i]

    local pos = item[KEY_POS]
    local colnocomputed = item[KEY_COLNO_COMPUTED]

    if pos == POS_LEFT then
      rightside = false
    elseif pos == POS_RIGHT then
      rightside = true
    elseif pos == POS_NEAREST then
      rightside = (item[KEY_XPOS] >= x_textmiddle[colnocomputed])
    else
```

`pos` must be POS_AUTO or POS_REVERSE

```
      rightside = RIGHTSIDE_LOOKUP_TABLE[recto][colnocomputed][pos]
    end

    local marginno = MARGINNO_LOOKUP_TABLE[recto][colnocomputed][rightside]

    if rightside then
      item[KEY_SIDE_COMPUTED] = 0
      item[KEY_XSHIFT_COMPUTED] = -item[KEY_XPOS]
                                  + x_textright[colnocomputed]
    else
      item[KEY_SIDE_COMPUTED] = 1
      item[KEY_XSHIFT_COMPUTED] = -item[KEY_XPOS]
                                  + x_textleft[colnocomputed]
    end
    item[KEY_MARGINNO_COMPUTED] = marginno

  end

end
```

(*End of definition for* `compute_items_horizontal`.)

`get_y_item_top` Return the *y*-coordinate of the top of the item described by `item_data`.

```
local function get_y_item_top(item_data)
  return item_data[KEY_YPOS]
         + item_data[KEY_YSHIFT_COMPUTED]
         + item_data[KEY_HEIGHT]
end
```

(*End of definition for* `get_y_item_top`.)

`get_y_item_bottom` Return the *y*-coordinate of the bottom of the item described by `item_data`.

```
local function get_y_item_bottom(item_data)
  return item_data[KEY_YPOS]
         - item_data[KEY_DEPTH]
```

```
          + item_data[KEY_YSHIFT_COMPUTED]
end
```

(*End of definition for* `get_y_item_bottom`*.*)

get_ysep_list

Calculate the separation to be used between adjacent marginal content items as described in `item_data_list`. The list is assumed to be sorted so that items are in the order they should appear on the page, top to bottom.

The idea is that we have the following arrangement for $i = 1, \ldots,$ `#item_data_list`:

```
  ⋮
item_data_list[i]
 ysep_list[i]
item_data_list[i+1]
  ⋮
```

Also set `ysep_list[0]` and `ysep_list[#item_data_list]` to 0, to avoid checking when these values are accessed (although they are not used).

```
local function get_ysep_list(item_data_list)

  local ysep_list = {}

  ysep_list[0] = 0
  for i=1,#item_data_list-1 do
    ysep_list[i] = math.max(
                     item_data_list[i][KEY_YSEP_BELOW],
                     item_data_list[i+1][KEY_YSEP_ABOVE]
                   )
  end
  ysep_list[#item_data_list] = 0

  return ysep_list

end
```

(*End of definition for* `get_ysep_list`*.*)

12.10 Computing vertical positions

12.10.1 Computing `optfixed` enabled

compute_items_vertical_optfixed_enabled

For every `item_data` in `item_data_list` describing an item of type `TYPE_OPTFIXED`, check for a clash with an item of type `TYPE_FIXED`. If so, set `item_data[KEY_ENABLED_-COMPUTED]` to `false`. Every item described in `item_data_list` is assumed to be on the same page and to have `KEY_YSHIFT` set to the default.

```
local function compute_items_vertical_optfixed_enabled(item_data_list)

  local optfixed_item_data_list = {}
  local fixed_item_data_list = {}

  for _,item_data in pairs(item_data_list) do
    if item_data[KEY_TYPE] == TYPE_OPTFIXED then
      optfixed_item_data_list[#optfixed_item_data_list+1] = item_data
    elseif item_data[KEY_TYPE] == TYPE_FIXED then
```

```
      fixed_item_data_list[#fixed_item_data_list+1] = item_data
    end
  end

  for _,optfixed_item_data in pairs(optfixed_item_data_list) do
    local optfixed_y_item_top = get_y_item_top(optfixed_item_data)
    local optfixed_y_item_bottom = get_y_item_bottom(optfixed_item_data)

    for _,fixed_item_data in pairs(fixed_item_data_list) do
      local fixed_y_item_top = get_y_item_top(fixed_item_data)
      local fixed_y_item_bottom = get_y_item_bottom(fixed_item_data)

      if (
        (
          (fixed_y_item_bottom - optfixed_y_item_top)
          <
          math.max(
            fixed_item_data[KEY_YSEP_BELOW],
            optfixed_item_data[KEY_YSEP_ABOVE]
          )
        )
        and
        (
          (optfixed_y_item_bottom - fixed_y_item_top)
          <
          math.max(
            optfixed_item_data[KEY_YSEP_BELOW],
            fixed_item_data[KEY_YSEP_ABOVE]
          )
        )
      ) then
        optfixed_item_data[KEY_ENABLED_COMPUTED] = false
        break
      end
    end
  end

end
```

(*End of definition for* `compute_items_vertical_optfixed_enabled`*.*)

12.10.2 Computing vertical adjustment

compute_items_vertical_adjustment For every `item_data` in `item_data_list`, compute the field relevant to vertical positioning, namely `KEY_YSHIFT_COMPUTED`, based on the layout information in `page_data`. Every item described in `item_data_list` is assumed to be on the same page and to have `KEY_YSHIFT` set to the default, and the list is assumed to be sorted so that items are in the order they should appear on the page, top to bottom.

```
local function compute_items_vertical_adjustment(item_data_list,page_data)
```

Immediately return if `item_data_list` is empty, to avoid edge cases

```
  if #item_data_list == 0 then
    return
  end

  local ysep_list = get_ysep_list(item_data_list)
```
First pass of computation (downward). `y_limit_above` will always be the highest *y*-coordinate at which the top of next item below can appear.

```
  local y_limit_above = (
    page_data[KEY_VOFFSET]
    + page_data[KEY_PAPERHEIGHT]
    - item_data_list[1][KEY_YSEP_PAGE_TOP]
  )

  for i=1,#item_data_list do
    local item_data = item_data_list[i]

    local y_item_top = get_y_item_top(item_data)

    if y_item_top > y_limit_above then
      if item_data[KEY_TYPE] == TYPE_NORMAL then
        item_data[KEY_YSHIFT_COMPUTED] = item_data[KEY_YSHIFT_COMPUTED]
                                         + (y_limit_above - y_item_top)
      end
    end

    y_limit_above = get_y_item_bottom(item_data) - ysep_list[i]
  end
```
Second pass of computation (upward). `y_limit_below` will always be the lowest *y*-coordinate at which the bottom of next item above can appear.

```
  local y_limit_below = (
    page_data[KEY_VOFFSET]
    + item_data_list[#item_data_list][KEY_YSEP_PAGE_BOTTOM]
  )

  for i=#item_data_list,1,-1 do
    local item_data = item_data_list[i]

    local y_item_bottom = get_y_item_bottom(item_data)

    if y_item_bottom < y_limit_below then
      if item_data[KEY_TYPE] == TYPE_NORMAL then
        item_data[KEY_YSHIFT_COMPUTED] = item_data[KEY_YSHIFT_COMPUTED]
                                         + (y_limit_below - y_item_bottom)
      end
    end

    y_limit_below = get_y_item_top(item_data) + ysep_list[i-1]
  end

end
```
(*End of definition for* `compute_items_vertical_adjustment`*.*)

12.10.3 Checking vertical adjustment

Messages to use when checking results of vertical adjustment.

```
local ITEM_PASSED_YSEP_PAGE_TOP_MESSAGES = {
  [TYPE_NORMAL] = 'Moveable item > ysep page top',
  [TYPE_FIXED] = 'Topmost fixed item > ysep page top',
  [TYPE_OPTFIXED] = 'Topmost optfixed item > ysep page top',
}
local ITEM_CLASH_MESSAGES = {
  [TYPE_NORMAL] = {
    [TYPE_NORMAL] = 'moveable items'
                    .. ' (this shouldn\'t happen)',
    [TYPE_FIXED] = 'moveable item above fixed item',
    [TYPE_OPTFIXED] = 'moveable item above optfixed item',
  },
  [TYPE_FIXED] = {
    [TYPE_NORMAL] = 'moveable item below fixed item',
    [TYPE_FIXED] = 'fixed items',
    [TYPE_OPTFIXED] = 'fixed item above optfixed item '
                      .. '(this shouldn\'t happen)',
  },
  [TYPE_OPTFIXED] = {
    [TYPE_NORMAL] = 'moveable items below optfixed item',
    [TYPE_FIXED] = 'fixed item below optfixed item '
                   .. '(this shouldn\'t happen)',
    [TYPE_OPTFIXED] = 'optfixed items '
                      .. '(this shouldn\'t happen)',
  },
}
local ITEM_PASSED_YSEP_PAGE_BOTTOM_MESSAGE = {
  [TYPE_NORMAL] = 'Moveable item < ysep page bottom',
  [TYPE_FIXED] = 'Bottommost fixed item < ysep page bottom',
  [TYPE_OPTFIXED] = 'Bottommost optfixed item < ysep page bottom',
}
```

`check_items_vertical` For the items described by the item_data in `item_data_list`, check whether any clash or fail to obey `ysep page top` or `ysep page bottom`. If so, write messages to `PROBLEM_REPORT_TABLE`.

```
local function check_items_vertical(item_data_list,page_data)
```

Immediately return if item_data_list is empty, to avoid edge cases

```
  if (#item_data_list) == 0 then
    return
  end

  local ysep_list = get_ysep_list(item_data_list)

  local item_data

```

If any item fails to obey `ysep page top`, the first one in the list does.

```
  item_data = item_data_list[1]
  if (
       get_y_item_top(item_data) > page_data[KEY_VOFFSET]
                                 + page_data[KEY_PAPERHEIGHT]
                                 - item_data[KEY_YSEP_PAGE_TOP]
  ) then
    table.insert(
      PROBLEM_REPORT_TABLE,
      get_data_page_number(item_data)
      .. ' ' .. ITEM_PASSED_YSEP_PAGE_TOP_MESSAGES[item_data[KEY_TYPE]]
    )
  end

  for i=2,#item_data_list do
    local item_data = item_data_list[i]
    local prev_item_data = item_data_list[i-1]
    if (
      get_y_item_top(item_data) > get_y_item_bottom(prev_item_data)
                                  - ysep_list[i-1]
    ) then
      table.insert(
        PROBLEM_REPORT_TABLE,
        get_data_page_number(item_data)
        .. ' Clash: ' ..
        ITEM_CLASH_MESSAGES[prev_item_data[KEY_TYPE]][item_data[KEY_TYPE]]
      )
    end
  end
```

If any item fails to obey `ysep page bottom`, the last one in the list does.

```
  item_data = item_data_list[#item_data_list]
  if (
    get_y_item_bottom(item_data) < page_data[KEY_VOFFSET]
                                   + item_data[KEY_YSEP_PAGE_BOTTOM]
  ) then
    table.insert(
      PROBLEM_REPORT_TABLE,
      get_data_page_number(item_data)
      .. ' ' .. ITEM_PASSED_YSEP_PAGE_BOTTOM_MESSAGE[item_data[KEY_TYPE]]
    )
  end

end
```

(*End of definition for* `check_items_vertical`.)

12.10.4 Core vertical position computation

`compute_items_vertical` For every `item_data` in `item_data_list`, compute the field relevant to vertical positioning, namely `KEY_YSHIFT_COMPUTED`, based on the layout information in `page_data`. This may involve setting the field `KEY_ENABLED_COMPUTED` to false. In such a case, the relevant item_data is removed from `item_data_list`.

```
local function compute_items_vertical(item_data_list,page_data)
```

Set `KEY_YSHIFT_COMPUTED` of each `item_data` to the user-supplied value.

```
  for i=1,#item_data_list do
    local item_data = item_data_list[i]

    item_data[KEY_YSHIFT_COMPUTED] = item_data[KEY_YSHIFT]
  end
```

Decide which items of type `ITEM_DATA_OPTFIXED` are to be disabled.

```
  compute_items_vertical_optfixed_enabled(item_data_list)
```

Strip any `item_data` with `KEY_ENABLED_COMPUTED` set to false from `item_data_list`.

```
  list_filter(item_data_list,function(item_data)
    return item_data[KEY_ENABLED_COMPUTED]
  end)
```

Sort `item_data_list` according to the stored position from top to bottom and left to right on the page, resolving ties using `KEY_ITEMNO`.

```
  table.sort(
    item_data_list,
    function(left,right)
      local y_diff = left[KEY_YPOS] - right[KEY_YPOS]

      if y_diff > 0 then
        return true
      elseif y_diff < 0 then
        return false
      end

      local x_diff = left[KEY_XPOS] - right[KEY_XPOS]

      if x_diff < 0 then
        return true
      elseif x_diff > 0 then
        return false
      end

      return (left[KEY_ITEMNO] < right[KEY_ITEMNO])
    end
  )

  compute_items_vertical_adjustment(item_data_list,page_data)

  check_items_vertical(item_data_list,page_data)

end
```

(*End of definition for* `compute_items_vertical`.)

`compute_items` For every item represented in `ITEM_DATA_MAIN_TABLE`, use the `page_data` stored in `PAGE_DATA_MAIN_TABLE` to compute the item_data values necessary to place the item correctly on the page, namely those indexed by: `KEY_COLNO_COMPUTED`, `KEY_XSHIFT_-COMPUTED`, `KEY_YSHIFT_COMPUTED`, `KEY_SIDE_COMPUTED`, `KEY_ENABLED_COMPUTED`.

```
local function compute_items()
```

Compute the maximum abspageno, which will be the last page of the document on which a item appears.

```
  local max_abspageno = 0

  for k,v in pairs(ITEM_DATA_MAIN_TABLE) do
    max_abspageno = math.max(v[KEY_ABSPAGENO],max_abspageno)
  end
```

list per_abspage_item_data_list will be a list indexed by absolute page numbers. Each entry will be a list (possibly empty) of item_data describing the items that appear on the corresponding page.

```
  local per_abspage_item_data_list = {}
```

Prepare per_abspage_item_data_list by making each entry an empty list, then fill it from ITEM_DATA_MAIN_TABLE.

```
  for i=1,max_abspageno do
    per_abspage_item_data_list[i] = {}
  end
  for _,item_data in pairs(ITEM_DATA_MAIN_TABLE) do
    local temp_table = per_abspage_item_data_list[item_data[KEY_ABSPAGENO]]
    temp_table[#temp_table+1] = item_data
  end
```

per_abspage_item_data_list will be a list indexed by abssolute page numbers. Each entry will be a page_data describing the corresponding page. Usually multiple entries will be the same page_data: in the loop, pagedatano will be the index of the last entry in PAGE_DATA_MAIN_TABLE with KEY_ABSPAGENO value less than or equal to abspageno. (There may be several such entries in PAGE_DATA_MAIN_TABLE because \marginalianewgeometry may have been called multiple times on the same page.) Note that PAGE_DATA_MAIN_TABLE[0] is available even if there was no data in the .aux file, because the defaults were stored by store_default_page_data.

```
  local per_abspage_page_data_list = {}
```

```
  local pagedatano = 0
  for abspageno = 1,max_abspageno do
```

```
    while (
      PAGE_DATA_MAIN_TABLE[pagedatano+1] ~= nil
      and
      PAGE_DATA_MAIN_TABLE[pagedatano+1][KEY_ABSPAGENO] == abspageno
    ) do
      pagedatano = pagedatano+1
    end
    per_abspage_page_data_list[abspageno] = PAGE_DATA_MAIN_TABLE[pagedatano]
  end
```

Iterate through all pages and perform the necessary computations.

```
  for abspageno=1,#per_abspage_item_data_list do
    local current_page_data = per_abspage_page_data_list[abspageno]
    local current_page_item_data_list = per_abspage_item_data_list[abspageno]
```

First, compute the horizontal positions, which includes sorting items into columns in two-column mode.

```
    compute_items_horizontal(current_page_item_data_list,current_page_data)
```

Sort the items into sublists corresponding to the margins in which they are located.

```
    local current_page_item_data_sublists = {}

    for i=0,5 do
      current_page_item_data_sublists[i] = {}
    end

    for _,item_data in pairs(current_page_item_data_list) do
      table.insert(
        current_page_item_data_sublists[item_data[KEY_MARGINNO_COMPUTED]],
        item_data
      )
    end
```

Compute vertical positons for each sublist.

```
    for i=0,5 do
      compute_items_vertical(
        current_page_item_data_sublists[i],
        current_page_data
      )
    end
  end
end
```

(*End of definition for* `compute_items`.)

12.11 Passing item_data back to LaTeX

`load_item_data` Set the relevant LaTeX counter and dimension variables to the values computed for `itemno`.

```
local function load_item_data(itemno)

  item = ITEM_DATA_MAIN_TABLE[tonumber(itemno)]
  if item == nil then
    item = ITEM_DATA_DEFAULTS
  end

  tex.count['l__marginalia_page_int'] = item[KEY_PAGENO]
  tex.count['l__marginalia_column_computed_int'] = item[KEY_COLNO_COMPUTED]
  tex.dimen['l__marginalia_xshift_computed_dim'] = item[KEY_XSHIFT_COMPUTED]
  tex.dimen['l__marginalia_yshift_computed_dim'] = item[KEY_YSHIFT_COMPUTED]
  tex.count['l__marginalia_side_computed_int'] = item[KEY_SIDE_COMPUTED]
  tex.count['l__marginalia_marginno_computed_int']
    = item[KEY_MARGINNO_COMPUTED]
  if item[KEY_ENABLED_COMPUTED] then
    tex.count['l__marginalia_enabled_computed_int'] = 1
  else
    tex.count['l__marginalia_enabled_computed_int'] = 0
  end

end
```

(*End of definition for* `load_item_data`.)

12.12 Export public functions

Finally, make available the functions that will be called from LaTeX using `\lua_now:n` and `\lua_now:e`.

```
return {
  store_default_page_data = store_default_page_data,
  store_page_data = store_page_data,
  check_page_data = check_page_data,

  store_item_data = store_item_data,
  check_item_data = check_item_data,

  compute_items = compute_items,

  load_item_data = load_item_data,

  write_problem_report = write_problem_report,

  write_page_change_report = write_page_change_report,
  write_item_change_report = write_item_change_report,
}
```

⟨/lua⟩

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.