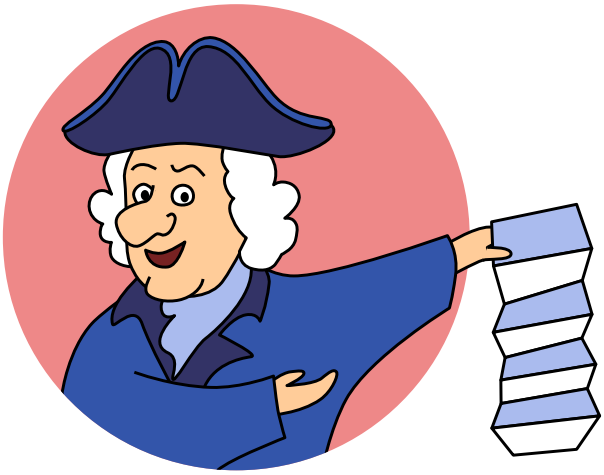


The `leporello` class

A simple LaTeX document class to create folded leaflets using columns and boxes

Jasper Habicht *

Version 1.0.1, released on 30 August 2025

---

1 Introduction

The `leporello` class is a simple LaTeX document class to create folded leaflets with the following key features:

- The document layout consists of a specific number of pages that can have varying widths and are placed next to each other. Each page contains one frame to contain typeset material.
- Material is typeset in boxes and boxes are positioned in columns which in turn are placed into the frames on the pages of the document.
- Columns are predefined and then placed into the frame of a page in the document layout. This way, columns can be used on pages with varying widths.
- Boxes can have padding (including bleed) and a background and are positioned inside a column.

A folded leaflet is sometimes called leporello in reference to the servant of Don Giovanni (known from the opera by Mozart) who, at one point, unfolds a lengthy zig-zag folded list of his master's love affairs.

There are some requirements for the creation of print data for high-quality folded leaflets: Especially if the leaflet is folded inward, the inner pages should be a bit less wide, so in general the pages of a leaflet are not of the exact same width (although the widths only differ slightly). Also, borderless printing requires that the print data adds a so-called bleed area added to the layout if the final document into which the typeset material extends. This area is trimmed off eventually to avoid

* E-mail: mail@jasperhabicht.de. Title image: © 2025 Hannah Klöber.

unprinted edges due to natural movement of the paper during guillotining. Crop marks need to be added to the print data showing the layout of the final document. This packages takes into account all these requirements while providing a lot of customization.

! This document class is still in an early development phase which means that bugs cannot be ruled out and some things may not work properly. Also, updates may be published frequently. The author is grateful for reporting any bugs via GitHub at https://github.com/jasperhabicht/leporello/issues. A site for asking questions about the document class and for suggestions for improvement is available at https://github.com/jasperhabicht/leporello/discussions.

2 Loading the document class

Use `\documentclass{leporello}` to load the document class. The document class loads the `graphicx` and the `l3draw` package.

The document class is based on the `article` class to enable basic typesetting commands. But due to the redefinition of the layout, certain functions of the `article` class, such as placing material into the header, the footer or the margin (which do not exist in the leporello layout) or using floats or footnotes won't work. Instead of floats, the document class provides a sophisticated alignment mechanism for boxes. Instead of footnotes, it provides its own note mechanism.

```
\leporelloset[‹prefix›]{‹options›}
```

Use `\leporelloset` or use the optional argument of the `\documentclass` command to globally set the document layout. The optional argument can be used to set a common prefix to all options. Options should be used without the `global/` prefix when used as option to the `\documentclass` command. The following options are available:

```
global/columns={‹list of dimensions›}
global/two columns
global/three columns
global/four columns
global/four columns wrap
global/five columns
global/six columns
```

Expects a comma-separated list of dimensions that describe the widths of the columns in the layout. If `columns` is not specified, the class assumes the default value of `97mm, 100mm, 100mm`. The width of the document layout is automatically calculated from the dimensions set via `columns`.

A few column layouts are predefined. These do not expect a value. The layouts will set the following list of dimensions:

Key	List of dimensions
`two columns`	99 mm, 99 mm
`three columns`	97 mm, 100 mm, 100 mm
`four columns`	99 mm, 99 mm, 99 mm, 100 mm
`four columns wrap`	97 mm, 99 mm, 100 mm, 101 mm
`five columns`	97 mm, 97 mm, 97 mm, 99 mm, 100 mm
`six columns`	99 mm, 99 mm, 96 mm, 96 mm, 100 mm, 100 mm

```
global/layout height={‹dimension›}
```

Expects a dimension that sets the height of the final layout. If `layout height` is not specified, the class assumes the default value of `210mm` (the height of A4 landscape).

`global/`**`auto typeset`**

If set, the columns are automatically typeset according to the order of the definition of the columns. This means that in a three-column layout for example, the first three defined columns will be placed on the first PDF page followed by the next three defined columns on the next PDF page. This key does not expect a value.

`global/`**`show frames`**

If set, frames around the single pages (columns) are shown. This key does not expect a value.

`global/`**`show ids`**

If set, the IDs of the boxes are shown. This key does not expect a value. The IDs of the boxes are integers starting from 1 and increasing in the order of typesetting of the boxes.

`global/`**`prepress`**

If set, the paper size is increased at all four edges by the width of the info area and crop marks are placed at the corners. This key does not expect a value.

`global/`**`info area`**`={‹dimension›}`

Expects a dimension for the width of the info area that is shown if `prepress` is set. If `info area` is not specified, the class assumes the default value of `10mm`.

`global/`**`bleed`**`={‹dimension›}`

Expects a dimension for the width of the bleed that is added to boxes with background and other material reaching to the edges of the document layout. If `bleed` is not specified, the class assumes the default value of `3mm`.

`\leporellobleed`

Retrieves the width of the bleed in pt.

`\leporelloboxwidth`

Retrieves the width of the current box including padding, but without bleed, in pt. The width of the box excluding padding can be retrieved with the `\linewidth` command.

3 Defining columns

Columns are grouped material to be typeset onto a specific page of the folded leaflet. A column can contain anything that can be placed in a TeX box in general, but the typical use of a column is to only serve as a frame to which boxes are attached. While the package provides a flow mechanism across boxes, no such mechanism is provided for columns. Thus, if a column contains more than fits into it, the relevant parts just spill over the frame edges.

```
\begin{leporellocolumn}{‹string›}
  ‹body›
\end{leporellocolumn}
```

A column is defined by the `leporellocolumn` environment which takes one mandatory argument taking the name of the column. A single dot (`.`) is reserved as name for an empty column.

4 Positioning boxes

```
\begin{leporellobox}[‹options›]
  ‹body›
\end{leporellobox}
```

Boxes can be positioned inside of columns. They can contain anything that can be placed in a TeX box in general. A box is defined by the `leporellobox` environment which takes one optional argument to set box-specific options. Options should be used without the `box/` prefix when used as option to the `leporellobox` environment. These are the following:

```
box/name={‹string›}
```

Expects a (unique) string denoting the name of the box which can be used for attaching boxes to each other or assigning boxes for the flow mechanism.

```
box/parent={‹string›}
```

Expects a string denoting the name of the box to which the current box is aligned. If not specified, the current column box is assumed as parent box. The parent box needs to be in the same column as the box that should be aligned to it.

```
box/align parent={‹tuple of poles›}
```

Expects a comma-separated list of two items (a tuple) which denotes the horizontal and vertical pole of which the intersection defines the coordinate of the parent box (per default this is the current column box) that serves as anchor to align the current box. If not specified, the default value `l, t` is assumed, denoting the top left corner. Available poles are:

Key	Meaning
`l`	left edge of the box
`hc`	horizontal center of the box
`r`	right edge of the box
`t`	top edge of the box
`vc`	vertical center of the box
`b`	bottom edge of the box

```
box/align self={‹tuple of poles›}
```

Expects a comma-separated list of two items (a tuple) which denotes the horizontal and vertical pole of which the intersection defines the coordinate of the current box that serves as anchor to align the current box to parent box (per default this is the current column box). If not specified, the

default value `l, t` is assumed, denoting the top left corner. Available poles are the same as for `align parent`.

```
box/offset={‹tuple of dimensions›}
```

Expects a comma-separated list of two dimensions (a tuple) which defines the offset of the anchor set via `align parent` and `align self`. The first dimension is the offset to the right, the second dimension the offset downwards. If not specified, the default value `0mm, 0mm` is assumed. Internally, the value is parsed as a floating point variable.

```
box/width={‹dimension›}
```

Expects a dimension to explicitly set the width of the current box. If not specified, the box is as wide as the parent column.

```
box/height={‹dimension›}
```

Expects a dimension to explicitly set the height of the current box. If not specified, the box takes its natural height which means that it is as high as necessary to fit the contents.

```
box/stretch
```

If set, the box is stretched until its relevant edge (the bottom edge if aligned at the top, the top edge if aligned at the bottom, in vertical typesetting the left or right edge) meets the edge of the parent column. This key does not expect a value.

This key can also help to align the relevant edges in cases where stretchable glue close to the edge would otherwise make the box overlap the edge of the parent column.

```
box/padding left={‹dimension›}
box/padding right={‹dimension›}
box/padding top={‹dimension›}
box/padding bottom={‹dimension›}
box/padding={‹key-value list›}
box/no padding
```

`padding left`, `padding right`, `padding top` and `padding bottom` each expect a dimension to describe the padding of the contents from the relevant edge of the box. If not specified, the default value of `7.5mm` is assumed.

All four padding settings can also be stated using the `padding` key and the subkeys `left`, `right`, `top`, `bottom`. Using this syntax, the default value of the padding setting would be expressed as `padding={left=7.5mm, right=7.5mm, top=7.5mm, bottom=7.5mm}`.

The key `no padding` sets all paddings to zero. This key does not expect a value.

```
box/pre={‹code›}
```

Expects a token list that is placed before the actual contents of the box. Should not contain typeset material. This can be handy, if multiple boxes should be styled in a similar way.

```
box/background color={‹color name›}
box/background color={none}
```

Expects a color name as defined via `l3color` or `none` which will not fill the background. If not specified, the default value of `none` is assumed.

`box/`**`background code`**`={‹code›}`

Expects typeset material that will be placed into the background of the box aligned at the upper left corner of the box. THe typeset material is clipped to the size of the box.

`box/`**`bleed`**`={‹list of values›}`

Expects a comma-separated list consisting of up to four items with the values `l` and `r`, `t` and `b` that describe the edges (left, right, top and bottom) where bleed should be added to the box. Note that bleed is never added to the inner edges where the columns meet.

`box/`**`store width`**`={‹control sequence›}`

Expects a single control sequence (macro) to store the width of the current box.

`box/`**`store height`**`={‹control sequence›}`

Expects a single control sequence (macro) to store the height of the current box.

`box/`**`flow into`**`={‹string›}`

Expects a string denoting the name of the box into which typeset material will flow into if it does not fit into the current box. The box to flow into can be in a following column or on a following page. See section 6 below.

5 Typesetting boxes into columns

`\leporellotypesetcolumns``[‹options›]{‹list of strings›}`

Using the command `\leporellotypesetcolumns` which takes one mandatory argument, previously defined columns can be placed onto a page of the document. The command expects as argument a comma-separated list of names of previously defined columns. These are then placed onto a document page from left to right while the width is taken from the setting via the `columns` key and the height is taken from the setting via the `layout height` key. With a `.`, an empty column can be added to the list.

The command takes one optional argument. Options should be used without the `typeset/` prefix when used as option to the `\leporellotypesetcolumns` command. The command accepts the following options:

`typeset/`**`reverse layout`**

If set, the widths of the columns are reversed, but the placement of the columns is still from left to right. This option does not take a value. The option should be used to typeset the verso of a folded leaflet that naturally has the widths of the columns reversed. This key does not expect a value.

`typeset/`**`reverse order`**

If set, the order of the columns is reversed, which means that the columns are typeset from right to left. Pagination is not affected. This key does not expect a value.

`typeset/`**`reverse pagination`**

If set, the pagination of the columns is reversed. This key does not expect a value.

`typeset/`**`continuous pagination`**

If set, columns are numbered continuously from left to right. Otherwise, the pagination starts on the last column of every odd shipout page, continues from left to right over the columns on the following shipout page and eventually from left ro right over the remaining columns on the first page. For example, in a document with three columns per shipout page the pagination would be 5, 6, 1 and 2, 3, 4 and then on the next two shipout pages 11, 12, 7 and 8, 9, 10. This key does not expect a value.

6 Flow mechanism

Typeset material can flow from one box to another box. To this end, the key `flow into` can be set to the relevant box and assigned the name of the box to flow into as value. The name of the relevant box can be set via the `name` key.

`\leporelloboxbreak`

The flow mechanism works across multiple boxes. But due to the way the typesetting mechanism of TeX works, it needs some manual adjustment if the typeset material flows across boxes of different width. In this case, the command `\leporelloboxbreak` or `\pagebreak` should be inserted at the point where the break should take place. The command `\leporelloboxbreak` is especially suited for justified typesetting as it will keep the last line before the break justified in contrast to the `\pagebreak` command.

Typeset material can only flow into boxes in the following column or page. If typeset material should flow backwards on the same page, this can be achieved using the key `reverse order` on the relevant page.

6.1 Right-to-left and vertical typesetting

The document class natively supports right-to-left and vertical typesetting using the mechanisms provided by LuaLaTeX. The `babel` package also makes use of the mechanisms for bidirectional typesetting provided by LuaLaTeX, which means that this package will work best with a combination of LuaLaTeX and `babel` for right-to-left or vertical typesetting.

If right-to-left typesetting is activated, the layout, order and pagination of the layout will be reversed automatically.

If the `bidi` package is used (which is used by the `polyglossia` package), the following addition to the preamble is needed to enable right-to-left typesetting:

```
\AddToHook{leporello/typeset/before}{\setLTR}
\AddToHook{leporello/box/begin}{\setRTL}
```

In general, it has to be made sure that only the inner boxes are affected by the right-to-left or vertical typesetting mechanism and not the overall layout.

Vertical typesetting will typically affect the measurements of the boxes. The package sets to true the boolean `\l_leporello_layout_vertical_ltr_bool` for vertical typesetting from left

to right (for example used to write traditional Mongolian). The package sets to true the boolean `\l_leporello_layout_vertical_rtl_bool` for vertical typesetting from right to left (for example used to write Chinese, Japanese or Korean).

7 Other settings

7.1 Inserting images

One way to insert images is via the `\includegraphics` command provided by the `graphicx` package. Using the `background code` key, images can be added to the background of a box.

\leporelloimage[‹options›]{‹file name›}

With the command `\leporelloimage` images that cover full boxes can be inserted. This command can be placed in a `leporellobox` with zero padding and it should only be used inside a `leporellobox` environment. The command has one mandatory argument that takes a relative path and file name to select the image to be inserted. It also has one optional argument to take options to be used without the `image/` prefix. The following options are available:

image/**clip width**={‹dimension›}
image/**clip height**={‹dimension›}

Expect a dimension depicting the width and the height of the boxed image.

image/**scale**={‹floating point number›}

Expects a floating point number depicting the scaling factor of the image. This factor will not affect the size of the box.

image/**width**={‹dimension›}
image/**height**={‹dimension›}

Expect a dimension to explicitly set the width and the height of the image. Setting `width` will override a `scale` value. Setting `height` will override a `width` or `scale` value. The aspect ratio of the original image will always be kept.

image/**offset**={‹tuple of dimensions›}

Expects a comma-separated list consisting of two values (tuple) that describe the offset of the image that will be positioned per default so that the upper left corner sits at the upper left corner of the box. A positive offset will shift the image in upper left direction. Internally, the value is parsed as a floating point variable.

image/**ignore padding**={‹list of values›}

Expects a comma-separated list consisting of up to four items with the values `l` and `r`, `t` and `b` that describe the edges (left, right, top and bottom) where the image should ignore the padding of the current box.

image/**fill bleed**

If set, the image will spread into the bleed. This will result in a shift of the image by the size of the bleed which may need to be accounted for using `offset`. This key does not expect a value.

7.2 Defining colors

The package uses the color model of the `l3color` module. The `xcolor` package is not supported. To provide a user interface to define and select colors, the commands `\leporellocolordefine` and `\leporellocolorselect` are defined.

\leporellocolordefine{‹string›}{‹color model›}{‹list of values›}

The command `\leporellocolordefine` takes three arguments, the first being the name of the color to be defined. The second argument takes the color model (for example `rgb` or `cmyk`) and the third argument takes the color values. For more information about which color models are supported, please refer to the documentation of the `l3color` module.

\leporellocolorselect{‹string›}

The command `\leporellocolorselect` takes the name of the previously defined color as argument. All following objects are affected by this color setting. To colorize only a few letters, use curly braces for grouping.

7.3 Defining styles

\leporellosetstyle[‹prefix›]{‹string›}{‹key-value list›}

In order to simplify the setting of recurring options to the `leporellobox` environment, it is possible to group several of these options as style via the `\leporellosetstyle` command which takes as first argument the name of the newly defined style and as second argument the releevant options (key-value pairs). The style can then be used like an option to any `leporellobox` environment. The optional argument can be used to set a common prefix for the style definition. The following two expressions are equivalent:

```
\leporellosetstyle[box]{align bottom}{
  align parent={l,b},
  align self={l,b}
}

\leporellosetstyle{box/align bottom}{
  box/align parent={l,b},
  box/align self={l,b}
}
```

7.4 Notes

Due to the layout specifications, this document class does not support footnotes. Instead, it provides its own note mechanism that allows to insert marks for notes and printing the notes combined as list.

\leporellonote[‹integer›]{‹code›}

The command `\leporellonote` increases the internal note counter and inserts a mark. It stores the actual note provided in the argument in an internal sequence. Using the optional argument, the counter for the mark can be set explicitly.

`\leporelloprintnotes`

The command `\leporelloprintnotes` prints all notes as list. Per default, this is an `enumerate` environment. It also empties the internal sequence to store notes.

Using the command `\leporelloset`, note marks and note lists can be styled. The following options are available:

notes/**mark cmd**={‹control sequence›}

Expects a control sequence that takes exactly one argument being the value of the counter of the note. The counter has the L3 integer type. To print the marks as Roman numerals in square brackets separated by a space, for example, the following settings could be used:

```
\NewDocumentCommand{\squaredroman}{m}{\,[\romannumeral #1]}
\leporellonotesset{
  mark cmd = {\squaredroman}
}
```

notes/**list style**={‹string›}

Expects a name of the list environment such as `enumerate` or `itemize`. Using the `enumitem` package, for example, a custom list can be created and styled individually.

7.5 Hooks

The package offers three pairs of hooks that are positioned at the start and end of columns and boxes allowing for inserting code.

The package uses the hook `begindocument/before` to insert layout-related settings among other things via the `geometry` package as well as the hook `shipout/foreground` to insert code to draw cropmarks.

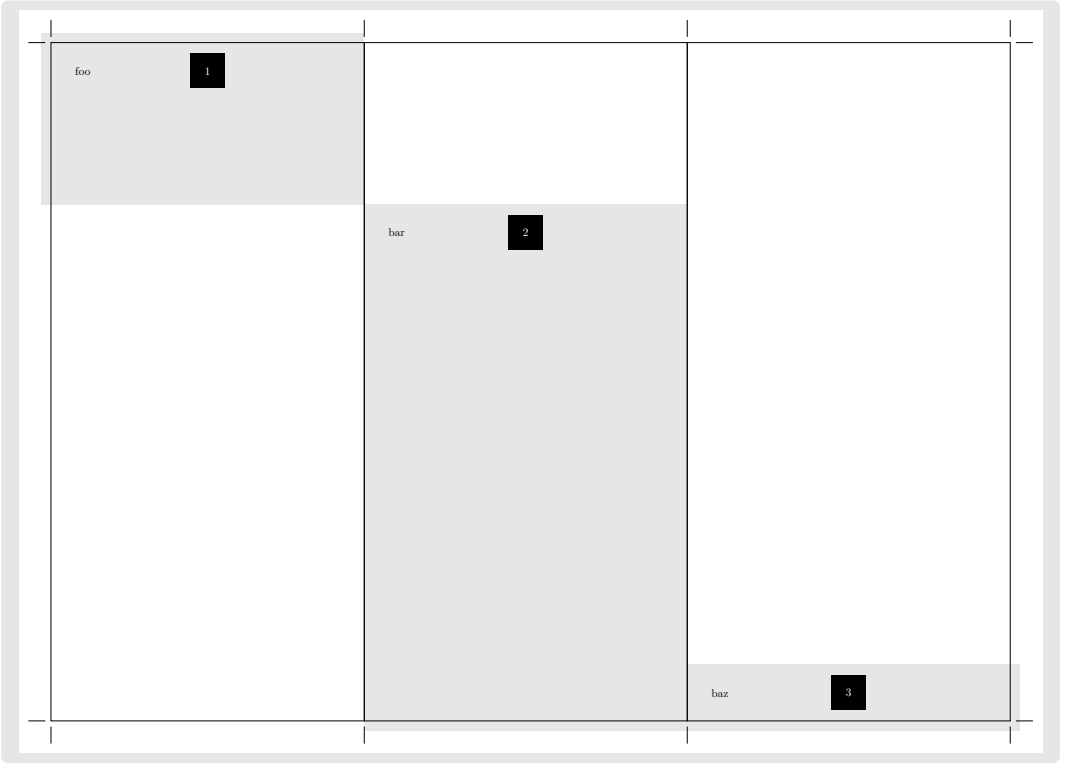
The package uses the hooks `leporello/typeset/before` and `leporello/box/begin` to unset and reset right-to-left and vertical typesetting mode.

Hook	Position
`leporello/column/begin`	before the code block stored via the `leporellocolumn` environment is typeset to the column box, preceded by a line of code that sets `\l_leporello_current_column_str` to the name of the current column
`leporello/column/end`	after the code block stored via the `leporellocolumn` environment is typeset to the column box
`leporello/box/begin`	before the code block stored via the `leporellobox` environment is typeset to the relevant box which is preceded by a line of code that sets `\l_leporello_current_box_int` to the ID of the current column and immediately followed by the code defined via the `pre` key
`leporello/box/end`	after the code block stored via the `leporellobox` environment is typeset to the relevant box

`leporello/typeset/before`	immediately after the start of the group that contains the typeset column
`leporello/typeset/after`	immediately before the end of the group that contains the typeset column

8 Example

The following example shows some of the basic ideas of the package by providing a code example and showing its output.



```
\documentclass[
  prepress,
  show frames,
  show ids
]{leporello}

\leporellosetstyle[box]{align bottom}{
  align parent={l,b},
  align self={l,b}
}

\begin{leporellocolumn}{example-a}
\begin{leporellobox}[
  background color=black!10,
  height=50mm,
  bleed={l,t}
]
foo
\end{leporellobox}
\end{leporellocolumn}

\begin{leporellocolumn}{example-b}
\begin{leporellobox}[
  background color=black!10,
  offset={0mm,50mm},
  stretch,
  bleed={b}
]
bar
\end{leporellobox}
\end{leporellocolumn}

\begin{leporellocolumn}{example-c}
\begin{leporellobox}[
  background color=black!10,
  align bottom,
  bleed={r,b}
]
baz
\end{leporellobox}
\end{leporellocolumn}

\begin{document}

\leporellotypesetcolumns{
  example-a,
  example-b,
  example-c
}

\end{document}
```

The example shows the use of the `prepress` key that adds cropmarks to the layout. It also shows how the `show frames` key adds frames around the columns. Finally, the `show box` key adds the box IDs to the boxes.

The default layout has three pages which are shown here. Three columns named `example-a`,

`example-b` and `example-c` are defined and contain one box each. A custom style is defined to combine the two keys needed to bottom-align the third box. A fixed height is set to for the first box and the second box has and offset and its height stretched to the bottom of the column. Bleed is added to the relevant edges of the three columns.

9 Changes

v0.7.0 (2025/08/08) First public beta release.

v0.8.0 (2025/08/12) Added box attachment mechanism and option to place arbitrary code to box background.

v0.9.0 (2025/08/15) Box attachment mechanism and flow mechanism use box name instead of ID. Enhanced pagination settings. Support for right-to-left and vertical typesetting.

v0.9.2 (2025/08/18) Various improvements and bug fixes.

v0.9.3 (2025/08/22) Stretch mechanism fixed.

v0.9.5 (2025/08/25) Improved alignment mechanism.

v1.0.0 (2025/08/27) Improved key management and flow mechanism. Added note mechanism.

v1.0.1 (2025/08/30) Various bug fixes.