LUA-WIDOW-CONTROL

Max Chernoff

v 3.0.0

---

ctan.org/pkg/lua-widow-control
github.com/gucci-on-fleek/lua-widow-control

Lua-widow-control is a Plain TEX/LATEX/ConTEXt/OpTEX package that removes widows and orphans without any user intervention. Using the power of LuaTEX, it does so *without* stretching any glue or shortening any pages or columns. Instead, lua-widow-control automatically lengthens a paragraph on a page or column where a widow or orphan would otherwise occur.

QUICK START

Ensure that your TEX Live/MikTEX distribution is up-to-date. Then, LATEX users just need to place `\usepackage{lua-widow-control}` in the preamble of their document. For more details, see the **Usage sections**.

CONTENTS

PRELIMINARIES

This manual begins with a brief introduction to widows, orphans, and lua-widow-control. For an extended introduction and discussion of these topics, please see the *Zpravodaj* article[1] distributed with this manual (Links: **local**, **CTAN**, **GitHub**). This same material is also available in two *TUGboat* articles[2,3] (Links: **local**, **CTAN**, **GitHub**) if you prefer. You can also skip ahead to the **installation instructions on page 7** or the **usage section starting at page 8**.

MOTIVATION

Unmodified TEX has only two familiar ways of dealing with widows and orphans: it can either shorten a page by one line, or it can stretch vertical whitespace. TEX was designed for mathematical and scientific typesetting, where a typical page has multiple section headings, tables, figures, and equations. For this style of document, TEX's default behaviour works quite well, since the slight stretching of whitespace between the various document elements is nearly imperceptible; however, for prose or other documents composed almost entirely of paragraphs, there is little vertical whitespace to stretch.

Lua-widow-control offers an alternative method of removing widows and orphans: instead of shortening a page or stretching vertical whitespace, lua-widow-control simply chooses a paragraph to lengthen by one line such that the widow or orphan is eliminated.

WIDOWS AND ORPHANS

Widows

A "widow" occurs when the majority of a paragraph is on one page or column, but the last line is on the following page or column. It not only looks quite odd for a lone line to be at the start of the page, but it makes a paragraph harder to read since the separation of a paragraph and its last line disconnects the two, causing the reader to lose context for the widowed line.

Orphans An "orphan" occurs when the first line of a paragraph is at the end of the page or column preceding the remainder of the paragraph. They are not as distracting for the reader, but they are still not ideal. Visually, widows and orphans are about equally disruptive; however, orphans tend not to decrease the legibility of a text as much as widows, so some authors choose to ignore them.

Broken Hyphens "Broken" hyphens occur whenever a page break occurs in a hyphenated word. These are not related to widows and orphans; however, breaking a word across two pages is at least as disruptive for the reader as widows and orphans. TEX identifies broken hyphens in the same ways as widows and orphans, so lua-widow-control treats broken hyphens in the same way.

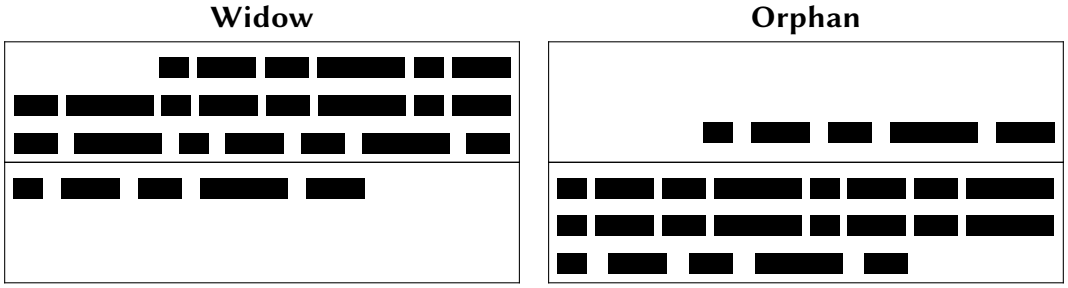

Figure 1 A visual comparison of widows and orphans.

TEX'S PAGINATION

Algorithm It is tricky to understand how lua-widow-control works if you aren't familiar with how TEX breaks pages and columns. For a full description, you should consult Chapter 15 of *The TEXBook*[4] ("How TEX Makes Lines into Pages"); however, this goes into much more detail than most users require, so here is a *very* simplified summary of TEX's page breaking algorithm:

TEX fills the page with lines and other objects until the next object will no longer fit. Once no more objects will fit, TEX will align the bottom of the last line with the bottom of the page by stretching any available vertical spaces if (in LATEX) `\flushbottom` is set; otherwise, it will break the page and leave the bottom empty.

However, some objects have "penalties" attached. Penalties encourage or discourage page breaks from occurring at specific places. For example, LATEX sets a negative penalty before section headings to encourage a page break there; conversely, it sets a positive penalty after section headings to discourage breaking.

To reduce widows and orphans, TEX sets weakly-positive penalties between the first and second lines of a paragraph to prevent orphans, and between the penultimate and final lines to prevent widows.

Behaviour Due to these "penalties" attached to widows and orphans, TEX tries to avoid creating them. Widows and orphans with small penalties attached—like LATEX's default values of 150—are only lightly coupled to the rest of the paragraph, while widows and orphans with large penalties—values of 10 000 or more—are treated as infinitely bad and are thus unbreakable. Intermediate values behave just as you would expect, discouraging page breaks proportional to their value.

However, when these lines are moved as a group, TEX will have to make a page or column with less lines. "**Demonstration**" goes into further detail about how TEX deals with these too-short pages or columns.

DEMONSTRATION

Although TEX's page breaking algorithm is reasonably straightforward, it can lead to complex behaviour when widows and orphans are involved. The usual choices, when rewriting is not possible, are to ignore them, stretch some glue, or shorten the page. **Table 1** has a visual comparison of these options, which we'll discuss in the following:

"Ignore" As you can see, the last line of the page is on a separate page from the rest of its paragraph, creating a widow. This is usually highly distracting for the reader, so it is best avoided for the reasons previously discussed.

"Shorten" This page did not leave any widows, but it did shorten the previous page by one line. Sometimes this is acceptable, but usually it looks bad because pages will then have different text-block heights. This can make the pages look quite uneven, especially when typesetting with columns or in a book with facing pages.

"Stretch" This page also has no widows and it has a flush bottom margin. However, the space between each pair of paragraphs had to be stretched.

If this page had many equations, headings, and other elements with natural space between them, the stretched out space would be much less noticeable. TEX was designed for mathematical typesetting, so it makes sense that this is its default behaviour. However, in a page with mostly text, these paragraph gaps look unsightly.

Also, this method is incompatible with grid typesetting, where all glue stretching must be quantised to the height of a line.

Ignore

Lua-widow-control can remove most
widows and orphans from a document,
without stretching any glue or shortening
any pages.
It does so by automatically lengthen-
ing a paragraph on a page where a widow or
orphan would otherwise occur. While TeX
breaks paragraphs into their natural length,
lua-widow-control is breaking the paragraph
1 line longer than its natural length. TeX's
paragraph is output to the page, but lua-
widow-control's paragraph is just stored for
later. When a widow or orphan occurs, lua-
widow-control can take over. It selects the
previously-saved paragraph with the least
badness; then, it replaces TeX's paragraph
with its saved paragraph. This increases
the text block height of the page by 1 line.
Now, the last line of the current page
can be pushed to the top of the next page.
This removes the widow or the orphan with-

out creating any additional work.

```
\parskip=0pt

\clubpenalty=0
\widowpenalty=0
```

Shorten

Lua-widow-control can remove most
widows and orphans from a document,
without stretching any glue or shortening
any pages.
It does so by automatically lengthen-
ing a paragraph on a page where a widow or
orphan would otherwise occur. While TeX
breaks paragraphs into their natural length,
lua-widow-control is breaking the paragraph
1 line longer than its natural length. TeX's
paragraph is output to the page, but lua-
widow-control's paragraph is just stored for
later. When a widow or orphan occurs, lua-
widow-control can take over. It selects the
previously-saved paragraph with the least
badness; then, it replaces TeX's paragraph
with its saved paragraph. This increases
the text block height of the page by 1 line.
Now, the last line of the current page
can be pushed to the top of the next page.

This removes the widow or the orphan with-
out creating any additional work.

```
\parskip=0pt

\clubpenalty=10000
\widowpenalty=10000
```

Stretch

Lua-widow-control can remove most
widows and orphans from a document,
without stretching any glue or shortening
any pages.

It does so by automatically lengthen-
ing a paragraph on a page where a widow or
orphan would otherwise occur. While TeX
breaks paragraphs into their natural length,
lua-widow-control is breaking the paragraph
1 line longer than its natural length. TeX's
paragraph is output to the page, but lua-
widow-control's paragraph is just stored for
later. When a widow or orphan occurs, lua-
widow-control can take over. It selects the
previously-saved paragraph with the least
badness; then, it replaces TeX's paragraph
with its saved paragraph. This increases
the text block height of the page by 1 line.

Now, the last line of the current page
can be pushed to the top of the next page.

This removes the widow or the orphan with-
out creating any additional work.

```
\parskip=0pt plus 1fill

\clubpenalty=10000
\widowpenalty=10000
```

Lua-widow-control

Lua-widow-control can remove most
widows and orphans from a document,
without stretching any glue or shortening
any pages.
It does so by automatically lengthen-
ing a paragraph on a page where a widow
or orphan would otherwise occur. While
TeX breaks paragraphs into their natural
length, lua-widow-control is breaking the
paragraph 1 line longer than its natural
length. TeX's paragraph is output to the
page, but lua-widow-control's paragraph is
just stored for later. When a widow or
orphan occurs, lua-widow-control can take
over. It selects the previously-saved para-
graph with the least badness; then, it re-
places TeX's paragraph with its saved para-
graph. This increases the text block height
of the page by 1 line.
Now, the last line of the current page
can be pushed to the top of the next page.

This removes the widow or the orphan with-
out creating any additional work.

```
\usepackage{lua-widow-control}
```

Table 1 A visual comparison of various automated widow handling techniques.

"lua-widow-control"

Lua-widow-control has none of these issues: it eliminates the widows in a document while keeping a flush bottom margin and constant paragraph spacing.

To do so, lua-widow-control lengthened the second paragraph by one line. If you look closely, you can see that this stretched the interword spaces. This stretching is noticeable when typesetting in a narrow text block, but is mostly imperceptible with larger widths.

Lua-widow-control automatically finds the "best" paragraph to stretch, so the increase in interword spaces should almost always be minimal.

INSTALLATION

Most up-to-date TEX Live and MikTEX systems should already have lua-widow-control installed. However, a manual installation may occasionally be required.

TEX Live

Run `tlmgr install lua-widow-control` in a terminal, or install using the "TEX Live Manager" GUI.

MikTEX

Run `mpm --install=lua-widow-control` in a terminal, or install using the "MikTEX Maintenance" GUI.

Manual

Currently, ConTEXt MkXL (LuaMetaTEX) users must manually install the package. Most other users will be better served by using the lua-widow-control supplied by TEX Live and MikTEX; however, all users may manually install the package if desired. The procedure should be fairly similar regardless of your OS, TEX distribution, or format.

Steps

1. Download `lua-widow-control.tds.zip` from **CTAN**, **GitHub** or the **ConTEXt Garden**.
2. Unzip the release into your `TEXMFLOCAL/` directory. (You can find its location by running `kpsewhich --var-value TEXMFHOME` in a terminal)
3. Refresh the filename database:
   - ConTEXt: `mtxrun --generate`
   - TEX Live: `mktexlsr`
   - MikTEX: `initexmf --update-fndb`

DEPENDENCIES

Lua-widow-control does have a few dependencies; however, these will almost cer-

tainly be met by all but the most minimal of TEX installations.

Plain TEX Lua-widow-control requires LuaTEX (≥ 0.85) and the most recent version of luatexbase (2015/10/04). Any version of TEX Live ≥ 2016 will meet these requirements.

LATEX Lua-widow-control requires LuaTEX (≥ 0.85), LATEX (≥ 2020/10/01), and microtype (any version). Any version of TEX Live ≥ 2021 will meet these requirements.

Lua-widow-control also supports a "legacy" mode for older LATEX kernels. This uses an older version of the LATEX code while still using the most recent Lua code. This mode requires LuaTEX (≥ 0.85), LATEX (≥ 2015/01/01), microtype (any version), and etoolbox (any version). Any version of TEX Live ≥ 2016 will meet these requirements.

Please note that when running in legacy mode, you cannot use the key–value interface. This legacy interface is undocumented, but mostly the same as the "Plain TEX" interface.

ConTEXt Lua-widow-control supports both ConTEXt MkXL (LuaMetaTEX) and ConTEXt MkIV (LuaTEX).

OpTEX Lua-widow-control works with any version of OpTEX and has no dependencies.

LuaMetaTEX Lua-widow-control has preliminary support for LuaMetaLATEX and LuaMetaPlain. All features should work identically to the LuaTEX-based version, although there are a few minor bugs. You should always make sure to use the latest engine, format, and lua-widow-control since these formats are under rapid development.

LOADING THE PACKAGE

Plain TEX	`\input lua-widow-control`
LATEX	`\usepackage{lua-widow-control}`
ConTEXt	`\usemodule[lua-widow-control]`
OpTEX	`\load[lua-widow-control]`

OPTIONS

Lua-widow-control is automatically enabled with the default settings as soon as you load it. Most users should not need to configure lua-widow-control; however, the packages provides a few commands.

Overview

LaTeX users can set the options either when loading the package (`\usepackage[⟨options⟩]{lua-widow-control}`) or at any point using `\lwcsetup{⟨options⟩}`.

ConTeXt users should use the `\setuplwc[⟨options⟩]` command for setting options at any point.

Plain TeX and OpTeX are a little different. Some options require you to set a register (i.e., `\lwcemergencystretch = ⟨dimension⟩`), while others use macro arguments (i.e., `\lwcnobreak{⟨option⟩}`).

Disabling

You may want to disable lua-widow-control for certain portions of your document. You can do so with the following commands:

Plain TeX/OpTeX	`\lwcdisable`
LaTeX	`\lwcsetup{disable}`
ConTeXt	`\setuplwc[state = stop]`

This prevents lua-widow-control from stretching any paragraphs that follow. If a page has earlier paragraphs where lua-widow-control was still enabled and a widow or orphan is detected, lua-widow-control will still attempt to remove the widow or orphan.

Enabling

Lua-widow-control is enabled as soon as the package is loaded. If you have previously disabled it, you will need to re-enable it to save new paragraphs.

Plain TeX/OpTeX	`\lwcenable`
LaTeX	`\lwcsetup{enable}`
ConTeXt	`\setuplwc[state = start]`

Automatically disabling

You may want to disable lua-widow-control for certain commands where stretching is undesirable such as section headings. Of course, manually disabling and then enabling lua-widow-control multiple times throughout a document would quickly become tedious, so lua-widow-control provides some options to do this automatically for you.

Lua-widow-control automatically patches the default LaTeX, ConTeXt, Plain TeX, OpTeX, memoir, KOMA-Script, and titlesec section commands, so you don't need to patch these. Any others, though, you'll need to patch yourself.

Plain TEX/OpTEX	`\lwcdisablecmd ⟨\macro⟩`
LATEX	`\lwcsetup{disablecmds = {⟨macronameone⟩, ⟨macronametwo⟩}}`
ConTEXt	`\prependtoks\lwc@patch@pre\to\everybefore⟨hook⟩`
	`\prependtoks\lwc@patch@pre\to\everyafter⟨hook⟩`

The Plain TEX, OpTEX, and ConTEXt commands *append* to the list of patched commands: they simply patch the provided commands while leaving the original patches in place. The LATEX option *sets* the list of patched commands: it replaces the default list with the provided list.

\emergencystretch

Lua-widow-control defaults to an `\emergencystretch` value of 3 em for stretched paragraphs, but you can configure this.

Lua-widow-control will only use the `\emergencystretch` when it cannot lengthen a paragraph in any other way, so it is fairly safe to set this to a large value. TEX accumulates badness when `\emergencystretch` is used, so it's pretty rare that a paragraph that requires any `\emergencystretch` will actually be used on the page.

Plain TEX/OpTEX	`\lwcemergencystretch = ⟨dimension⟩`
LATEX	`\lwcsetup{emergencystretch = ⟨dimension⟩}`
ConTEXt	`\setuplwc[emergencystretch = ⟨dimension⟩]`

Penalties

You can also manually adjust the penalties that TEX assigns to widows and orphans. Usually, the defaults are fine, but there are a few circumstances where you may want to change them.

Plain TEX/OpTEX	`\widowpenalty = ⟨integer⟩`
	`\clubpenalty = ⟨integer⟩`
	`\brokenpenalty = ⟨integer⟩`
LATEX	`\lwcsetup{widowpenalty = ⟨integer⟩}`
	`\lwcsetup{orphanpenalty = ⟨integer⟩}`
	`\lwcsetup{brokenpenalty = ⟨integer⟩}`
ConTEXt	`\setuplwc[widowpenalty = ⟨integer⟩]`
	`\setuplwc[orphanpenalty = ⟨integer⟩]`
	`\setuplwc[brokenpenalty = ⟨integer⟩]`

The value of these penalties determines how much TEX should attempt to stretch glue before passing the widow or orphan to lua-widow-control. If you set the values to 1 (default), TEX will stretch nothing and immediately trigger lua-widow-control; if you set the values to 10 000, TEX will stretch infinitely and lua-widow-control will never be triggered. If you set the value to some intermediate number, TEX will first attempt to stretch some glue to remove the widow or orphan; only if it fails will lua-widow-control come in and lengthen a paragraph. As a special case, if you set the values to 0, both TEX and lua-widow-control will completely ignore the widow or orphan.

Lua-widow-control will pick up on the values of `\widowpenalty`, `\clubpenalty`, and `\brokenpenalty` regardless of how you set them, so the use of these dedicated keys is entirely optional.

`\nobreak` Behaviour

When lua-widow-control encounters an orphan, it removes it by moving the orphaned line to the next page. The majority of the time, this is an appropriate solution. However, if the orphan is immediately preceded by a section heading (or `\nobreak`/`\penalty 10000`), lua-widow-control would naïvely separate a section heading from the paragraph that follows. This is almost always undesirable, so lua-widow-control provides some options to configure this.

Plain TEX/OpTEX	`\lwcnobreak{`⟨*value*⟩`}`
LATEX	`\lwcsetup{nobreak = `⟨*value*⟩`}`
ConTEXt	`\setuplwc[nobreak = `⟨*value*⟩`]`

The default value, `keep`, *keep*s the section heading with the orphan by moving both to the next page. The advantage to this option is that it removes the orphan and retains any `\nobreak`s; the disadvantage is that moving the section heading can create a large blank space at the end of the page.

The value `split` *split*s up the section heading and the orphan by moving the orphan to the next page while leaving the heading behind. This is usually a bad idea, but exists for the sake of flexibility.

The value `warn` causes lua-widow-control to give up on the page and do nothing, leaving an orphaned line. Lua-widow-control *warn*s the user so that they can manually remove the orphan.

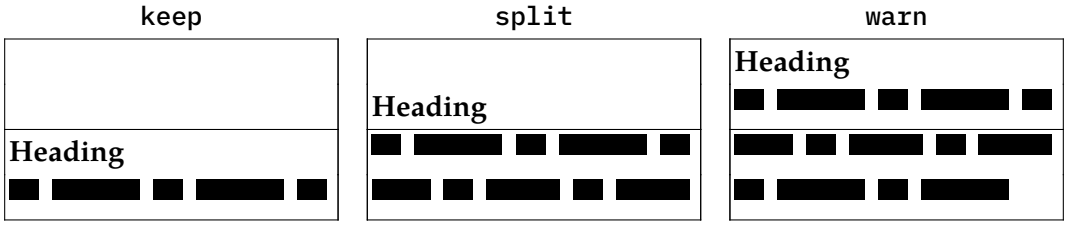

Figure 2 A visual comparison of the various `nobreak` options, where each box represents a different page.

Maximum Cost

Lua-widow-control ranks each paragraph on the page by how much it would "cost" to lengthen that paragraph. By default, lua-widow-control selects the paragraph on the page with the lowest cost; however, you can configure it to only select paragraphs below a selected cost.

If there aren't any paragraphs below the set threshold, then lua-widow-control won't remove the widow or orphan and will instead issue a warning.

Plain TEX/OpTEX	`\lwcmaxcost = ⟨integer⟩`
LATEX	`\lwcsetup{max-cost = ⟨integer⟩}`
ConTEXt	`\setuplwc[maxcost = ⟨integer⟩]`

Based on my testing, `max-cost` values less than 1 000 cause completely imperceptible changes in interword spacing; values less than 5 000 are only noticeable if you are specifically trying to pick out the expanded paragraph on the page; values less than 15 000 are typically acceptable; and larger values may become distracting. Lua-widow-control defaults to an infinite `max-cost`, although the "strict" and "balanced" modes sets the values to 5 000 and 10 000 respectively.

Lua-widow-control uses a "cost function" C that is initially defined as

$$C = \frac{d}{\sqrt{l}}$$

where d is the total demerits of the paragraph, and l is the number of lines in the paragraph.

Draft Mode

You can use the draft mode to visualize how lua-widow-control processes pages. Any remaining widows and orphans will be coloured red, any expanded paragraphs will be coloured green, and any lines moved to the next page will be coloured blue. In addition, the cost of each paragraph is shown in the right margin as it is here. 1846

Plain TEX/OpTEX	`\lwcdraft 1`
LATEX	`\lwcsetup{draft}`
ConTEXt	`\setuplwc[draft = start]`

1846

Advanced users may also customize the colours used by modifying the **`lwc.colours`** table. The table keys are currently `expanded`, `failure`, `moved`, and `cost`. The table values are RGB 3-tuples, where each element is a float between zero and one. infinite

You can also show or hide the costs and colours independently of each other. 8895

Plain TEX/OpTEX	`\lwcshowcosts 1`
	`\lwcshowcolours 1`
LATEX	`\lwcsetup{showcosts = true}`
	`\lwcsetup{showcolours = true}`
ConTEXt	`\setuplwc[showcosts = start]`
	`\setuplwc[showcolours = start]`

8895

Plain TEX users will need to modify their output routines to be able to see the costs. Before you ship out the page, you should call `\lwcpreshipout` with a box number as its argument. For example, here is some sample code to correctly reproduce the standard Plain TEX output routine: 57175

```
\newbox\tempshipoutbox                                  infinite
\output={                                               infinite
    \setbox\tempshipoutbox=\vbox{                       infinite
        \makeheadline                                   infinite
        \pagebody                                       infinite
        \makefootline                                   infinite
    }                                                   infinite
    \lwcpreshipout\the\tempshipoutbox                   infinite
    \shipout\box\tempshipoutbox                         infinite
    \advancepageno                                      infinite
}                                                       infinite
```

Setting the output routine like this is automatic in all other formats. Lua-widow-control does not modify the output routine in Plain TEX since most of its users likely have their own output routines.

Draft Offset

In draft mode, the paragraph costs are printed in the margins, offset a certain distance from the page edge. By default, this offset is set to 1 inch; however, you can change this to any value that you want:

Plain TEX/OpTEX	`\lwcdraftoffset = ⟨dimension⟩`
LATEX	`\lwcsetup{draftoffset = ⟨dimension⟩}`
ConTEXt	`\setuplwc[draftoffset = ⟨dimension⟩]`

Debug Mode

Lua-widow-control offers a "debug" mode that prints extra information in the log files. This may be helpful to understand how lua-widow-control is processing paragraphs and pages, although the information is likely inscrutable unless you are the package's author. If you are reporting an issue with lua-widow-control, make sure to compile your document with debug mode enabled!

Plain TEX/OpTEX	`\lwcdebug 1`
	`\lwcdebug 0`
LATEX	`\lwcsetup{debug = true}`
	`\lwcsetup{debug = false}`
ConTEXt	`\setuplwc[debug = start]`
	`\setuplwc[debug = stop]`

PRESETS

As you can see, lua-widow-control provides quite a few options. Luckily, there are a few presets that you can use to set multiple options at once. These presets are a good starting point for most documents, and you can always manually override individual options.

Presets are available only with LATEX and ConTEXt.

LATEX	`\lwcsetup{⟨preset⟩}`
ConTEXt	`\setuplwc[⟨preset⟩]`

`default` If you use lua-widow-control without any options, it defaults to this preset. In default mode, lua-widow-control takes all possible measures to remove widows and orphans and will not attempt to stretch any vertical glue. This usually removes $>95\%$ of all possible widows and orphans. The catch here is that this mode is quite aggressive, so it often leaves behind some fairly "spacey" paragraphs.

This mode is good if you want to remove (nearly) all widows and orphans from your document, without fine-tuning the results.

`strict` Lua-widow-control also offers a strict mode. This greatly restricts lua-widow-control's tolerance and makes it so that it will only lengthen paragraphs where the change will be imperceptible.

The caveat with strict mode is that—depending on the document— lua-widow-control will be able to remove less than a third of the widows and orphans. For the widows and orphans that can't be automatically removed, a warning will be printed to your terminal and log file so that a human can manually fix the situation.

This mode is good if you want the best possible typesetting and are willing to do some manual editing.

`balanced` Balanced mode sits somewhere between default mode and strict mode. This mode first lets TEX stretch a little glue to remove the widow or orphan; only if that fails will it then trigger lua-widow-control. Even then, the maximum paragraph cost is capped. Here, lua-widow-control can usually remove 90% of a document's potential widows and orphans, and it does so while making a minimal visual impact.

This mode is recommended for most users who care about their document's typography. This mode is not the default since it doesn't remove all widows and orphans: it still requires a little manual intervention.

LUA INTERFACE

Lua-widow-control provides a few public functions and tables that you can safely use or modify as documented. Any Lua interfaces that aren't documented below are subject to change at any time, but ff you do want to use an undocumented interface, please let me know and can easily document it.

Option	default	balanced	strict
max-cost	∞	10000	5000
emergencystretch	3em	1em	0pt
nobreak	keep	keep	warn
widowpenalty	1	500	1
orphanpenalty	1	500	1
brokenpenalty	1	500	1

Table 2 Lua-widow-control options set by each mode.

Costs

Lua-widow-control uses a "cost function" to select which paragraph to expand. Typically, this depends on the number of lines and demerits in the broken paragraph; however, you can redefine this function to do something else if you want. The default function is defined as:

```
--- The "cost function" to use.
---
--- @param demerits number The demerits of the broken paragraph
--- @param lines number The number of lines in the broken paragraph
--- @param nat_demerits number The demerits of the naturally-broken paragraph
--- @param nat_lines number The number of lines in the naturally-broken paragraph
--- @param head node The head of the broken paragraph
--- @return number cost The cost of the broken paragraph
function lwc.paragraph_cost(demerits, lines, nat_demerits, nat_lines,
head)
    return demerits / math.sqrt(lines)
end
```

Activation

Typically, lua-widow-control determines if there are widows and orphans at the end of a page by checking the `\outputpenalty` register. However, you can use a custom check if you want. The default check is defined as:

```
--- Determines if we should "activate" lwc for the current page/column.
---
--- @param penalty number The \outputpenalty for the current page/column
```

```
--- @param paragraphs table<table<string, node|number>> The `paragraphs` table
--- @param head node The head of the current page/column
--- @return boolean activate True if lwc should move the last line on this page
function lwc.should_remove_widows(penalty, paragraphs, head)
    return is_matching_penalty(penalty)
end
```

By setting a custom activation and cost function, you can transform lua-widow-control from a widow and orphan remover into a custom layout customization tool.

COMPATIBILITY

The lua-widow-control implementation is almost entirely in Lua, with only a minimal TEX footprint. It doesn't modify the output routine, `\everypar`, and it doesn't insert any whatsits. This means that it should be compatible with nearly any TEX package, class, and format. Most changes that lua-widow-control makes are not observable on the TEX side.

However, on the Lua side, lua-widow-control modifies much of a page's internal structure. This should not affect any TEX code; however, it may surprise Lua code that modifies or depends on the page's low-level structure. This does not matter for Plain TEX or LATEX, where even most Lua-based packages don't depend on the node list structure; nevertheless, there are a few issues with ConTEXt.

However, on the Lua side, lua-widow-control modifies much of a page's internal structure. This should not affect any TEX code; however, it may surprise Lua code that modifies or depends on the page's low-level structure. This does not affect Plain TEX or LATEX where even most Lua-based packages don't depend on the node list structure. ConTEXt *does* depend on this internal node structure; however, I have carefully tested the package to ensure that this causes no issues.

Finally, keep in mind that adding lua-widow-control to a document will almost certainly change its page break locations.

Formats

Lua-widow-control runs on all known LuaTEX-based formats: Plain LuaTEX, LuaLATEX, ConTEXt MkIV, and OpTEX. Unless otherwise documented, all features should work equally well in all formats.

Lua-widow-control is also fully-compatible with the LuaMetaTEX-based formats: ConTEXt MkXL/LMTX, LuaMetaLATEX, and LuaMetaPlain. ConTEXt MkXL works equally well as ConTEXt MkIV and LuaLATEX; however, LuaMetaLATEX and

LuaMetaPlain support is still quite early. All features should work, although there are still a few minor bugs.

All told, lua-widow-control supports 7 different format/engine combinations.

Columns

Since TEX and the formats implement column breaking and page breaking through the same internal mechanisms, lua-widow-control removes widows and orphans between columns just as it does with widows and orphans between pages.

Lua-widow-control is known to work with the LATEX class option `twocolumn` and the two-column output routine from Chapter 23 of *The TEXBook*[4].

Performance

Lua-widow-control runs entirely in a single pass, without depending on any `.aux` files or the like. Thus, it shouldn't meaningfully increase compile times. Although lua-widow-control internally breaks each paragraph twice, modern computers break paragraphs near-instantaneously, so you are not likely to notice any slowdown.

ε-TEX penalties

Knuth's original TEX has three basic line penalties: `\interlinepenalty`, which is inserted between all lines; `\clubpenalty`, which is inserted after the first line; and `\widowpenalty`, which is inserted before the last line. The ε-TEX extensions generalize these commands with a syntax similar to `\parshape`: with `\widowpenalties` you can set the penalty between the last, second last, and *n*th last lines of a paragraph; `\interlinepenalties` and `\clubpenalties` behave similarly.

Lua-widow-control makes no explicit attempts to support these new `-penalties` commands. Specifically, if you give a line a penalty that matches either `\widowpenalty` or `\clubpenalty`, lua-widow-control will treat the lines exactly as it would a widow or orphan. So while these commands won't break lua-widow-control, they are likely to lead to some unexpected behaviour.

Grids

Lua-widow-control is fully compatible with the grid snapping features of ConTEXt ConTEXt MkIV and ConTEXt MkXL.

Footnotes

If there are footnotes (or any other type of inline `\insert`) present in the moved line, lua-widow-control will move both the "footnote mark" and the "footnote text" such that both are on the same page. However, this may lead to an odd blank space at the bottom of the page since lua-widow-control needs to move both the line and its footnotes. Footnotes cause the same page-breaking issues in unmodified Plain TEX and LATEX, so this is mostly unavoidable.

STABILITY

The documented interfaces of lua-widow-control can be considered stable: I'm not planning on removing or modifying any existing options or commands in any way that would break documents.

However, lua-widow-control's page breaking *is* subject to change. I will attempt to keep page breaks the same whenever reasonable; however, I will *rarely* make modifications to the algorithm when I can improve the output quality. Any such changes will be clearly noted in the release notes.

SHORT LAST LINES

When lengthening a paragraph with `\looseness`, it is common advice to insert ties (~) between the last few words of the paragraph to avoid overly-short last lines[4]. Lua-widow-control does this automatically, but instead of using ties or `\hbox`es, it uses the `\parfillskip` parameter. When lengthening a paragraph (and only when lengthening a paragraph—remember, lua-widow-control doesn't interfere with TEX's output unless it detects a widow or orphan), lua-widow-control sets `\parfillskip` to `0.75\hsize plus 0.05\hsize minus 0.75\hsize`. This normally makes the last line of a paragraph be at least 20% of the overall paragraph's width, thus preventing ultra-short lines.

KNOWN ISSUES

- When a three-line paragraph is at the end of a page forming a widow, lua-widow-control will remove the widow; however, it will leave an orphan. This issue is inherent to any process that removes widows through paragraph expansion and is thus unavoidable. Orphans are considered to be better than widows[5], so this is still an improvement.
- Lua-widow-control only attempts to expand paragraphs; it never attempts to shrink them. See the *TUGboat* article`<article>` §15.3 for further discussion. **(Issue #33)**
- Lua-widow-control can only expand paragraphs that fit completely on a page. This is unavoidable due to the one-page-at-a-time model: you can't modify the bottom half of a paragraph since its top half has already shipped out, and you can't expand the top half of a paragraph since that can't remove orphans. This

only causes issues if your document has paragraphs so long that a page only has two half-paragraphs and zero whole paragraphs.

- Sometimes a widow or orphan cannot be eliminated because no paragraph has enough stretch. Sometimes this can be remediated by increasing lua-widow-control's `\emergencystretch`; however, some pages just don't have any suitable paragraph.

  Long paragraphs with short words tend to be stretchier than short paragraphs with long words since these long paragraphs have more interword glue. Narrow columns also stretch more easily than wide columns since you need to expand a paragraph by less to make a new line.

- Lua-widow-control only attempts to expand paragraphs on a page with a widow or orphan. A global system like in *A general framework for globally optimized pagination*[6] would solve this; however, this is both NP-complete[7] and impossible to solve in a single pass. Very rarely would such a system remove widows or orphans that lua-widow-control cannot.

CONTRIBUTIONS

If you have any issues with lua-widow-control, please create an issue at the **project's GitHub page**. Or, if you think that you can solve any of the "**Known Issues**" or add any new features, **submit a PR**. Thanks!

LICENSE

REFERENCES

1. Chernoff, M (2022a). Automatically removing widows and orphans with lua-widow-control. *Zpravodaj Československého sdružení uživatelů TeXu*, *2022*(1–4), 49–76. DOI: 10.5300/2022-1-4/49
2. Chernoff, M (2022b). Automatically removing widows and orphans with `lua-widow-control`. *TUGboat*, *43*(1), 28–39. DOI: 10.47397/tb/43-1/tb133chernoff-widows
3. Chernoff, M (2022c). Updates to lua-widow-control. *TUGboat*, *43*(3), 340–342. DOI: 10.47397/tb/43-3/tb135chernoff-lwc
4. Knuth, DE (2020). *The TEXBook*. Addison–Wesley. ctan.org/pkg/texbook
5. Bringhurst, R (2004). *The Elements of Typographic Style*. (3rd ed.). Hartley & Marks.
6. Mittelbach, F (2018). A general framework for globally optimized pagination. *Computational Intelligence*, *35*(2), 242–284. DOI: 10.1111/coin.12165
7. Plass, MF (1981). *Optimal pagination techniques for automatic typesetting systems*. (PhD thesis). Stanford University. tug.org/docs/plass/plass-thesis.pdf

CHANGELOG

All notable changes to lua-widow-control will be listed here, in reverse chronological order. **Changes listed in bold** are important changes: they either remove options or commands, or may change the location of page breaks.

v3.0.0 (2022-11-22)

- Add the new *TUGboat* and *Zpravodaj* articles.
- Add and document the public Lua interfaces.
- Change `\parfillskip` settings for lengthened paragraphs to more strongly prevent short last lines. **May affect page breaks.**
- Add the ability to configure the horizontal offset for the paragraph costs printed in draft mode.
- Add support for **LuaMetaLaTeX and LuaMetaPlain**. All features should work identically to the LuaTeX-based version, although there are a few minor bugs. (#40)
- Fully support inserts/footnotes in LuaMetaTeX (#38).
- Add support for presets in ConTeXt.
- Add support for node colouring in ConTeXt and OpTeX (#39).

v2.2.2 (2022-08-23)

- Add preliminary support for inserts/footnotes in LuaMetaTeX (#38).
- Use the built-in LaTeX key–value interface where available.

  This means that lua-widow-control now also reads the global class options.

- Add support for split footnotes (#37).

v2.2.1 (2022-07-28)

- Fix crashes with recent LuaMetaTeX (ConTeXt MkXL). See also **this thread**.
- No longer show "left parfill skip" warnings with ConTeXt LMTX/MkXL (#7).

v2.2.0 (2022-06-17)

- Fix paragraphs not being properly saved for potential expansion. **May affect page breaks.**
- Add a new `draft` option (**#36**).
- Fix a node memory leak (**#29**). You should now be able to use lua-widow-control on documents with > 10 000 pages.
- Use `\lua_load_module:n` when available.
- Add additional metadata to the documentation.

v2.1.2 (2022-05-27)

- Fully-support footnotes/inserts: lua-widow-control now moves the "footnote text" with the "footnote mark" when it moves a line to the next page.
- No longer attempt to expand paragraphs in `\vboxes`
- Minor documentation updates

v2.1.1 (2022-05-20)

- Prevent spurious `under/overfull \vbox` warnings when widows/orphans are removed
- Add TUGboat article to the distributed documentation
- Rewrite many portions of the manual
- Add support for `luahbtex` and `mmoptex` (**#35 @vlasakm**)
- Fix the (undocumented) `microtype` LaTeX option

v2.1.0 (2022-05-14)

- Fully support grid snapping in ConTeXt
- New warnings when a new widow/orphan is inadvertently created
- Significant internal rewrites
- Add Plain and OpTeX interfaces to `\nobreak` behaviour and debug mode

v2.0.6 (2022-04-23)

- Emergency fix for renamed LMTX engine Lua functions
- Internal LaTeX refactoring

v2.0.5 (2022-04-13)

- Support nested `\lwcdisablecmd` macros
- Fix `\lwcdisablecmd` in Plain TeX
- Support command patching in OpTeX
- Patch memoir to prevent spurious asterisks at broken two-line paragraphs (#32)

v2.0.4 (2022-04-07)

- Don't expand paragraphs typeset during output routines (#31)

v2.0.3 (2022-03-28)

- Automatically patch section commands provided by memoir, KOMA-Script, and titlesec.

v2.0.2 (2022-03-20)

Final release present in TeX Live 2021

- Add `balanced` mode preset.

v2.0.1 (2022-03-18)

- Documentation updates (#25)
- Bug fixes (#28)

v2.0.0 (2022-03-07)

- **Page breaks may be slightly different**
- **Removed `\lwcemergencystretch` and `\lwcdisablecmd` in LaTeX. Please use the new key–value interface**
- Use expl3 for the LaTeX files (#20)
- Use a key–value interface for configuration with LaTeX (#11)
- Silence some extraneous `luatexbase` info messages
- Add a "debug mode" to print extra information (#12)
- Fix error message line wrapping
- Don't reset `\interlinepenalty` and `\brokenpenalty`
- Set and analyze `\displaywidowpenalty`
- Keep section headings with moved orphans (#17)
- Add the ability to configure the maximum paragraph cost (#22)

- Add a "strict" mode
- Use an improved cost function to select the best paragraph to lengthen (#23)
- Dozens of bug fixes
- Miscellaneous documentation updates

v1.1.6 (2022-02-22)

- Add support for the OpTeX format/macro package.
- Add support for the LuaTeX/MKIV version of ConTeXt.
- Ensure compatibility with the new `linebreaker` package.
- Fix a small bug with `\lwcdisablecmd`.
- Test the `\outputpenalty` for the specific values that indicate a widow or orphan.

v1.1.5 (2022-02-15)

- Improve the appearance of the demo table in the documentation (#4)
- Improve compatibility with microtype
- Block loading the package without LuaTeX
- Improve logging
- Bug fix to prevent crashing

v1.1.4 (2022-02-04)

- Enable protrusion/expansion in the demo table in the documentation (#3)
- Fix `\prevdepth` bug

v1.1.3 (2022-01-30)

- Fix bug when used with the LaTeX `calc` package. (#2)

v1.1.2 (2021-12-14)

- Fix crash under ConTeXt LMTX

v1.1.1 (2021-11-26)

- Minor documentation updates

v1.1.0 (2021-11-08)

- Extensive documentation updates
- Clarify that lua-widow-control *does* in fact support columns

- Add `\lwcdisablecmd` macro to disable lua-widow-control for certain commands
- Automatically disable lua-widow-control inside section headings (uses `\lwcdisablecmd`)
- Add automated tests to prevent regressions
- Fix a rare issue that would cause indefinite hangs

v1.0.0 (2021-10-09)

Initial release

IMPLEMENTATION

From here and until the end of this manual is the raw source code of lua-widow-control. This is primarily of interest to developers; most users need not read further.

This code vaguely resembles the typical LaTeX literate programming style, although I use extensive inline comments instead of arcane docstrip macros. Hopefully this is useful as a reference for advanced lua-widow-control users as well as anyone doing extensive node manipulation in LuaTeX.

If want to offer any improvements to the code below, please open an issue or a PR on **GitHub**.

lua-widow-control.lua

```
--[[
    lua-widow-control
    https://github.com/gucci-on-fleek/lua-widow-control
    SPDX-License-Identifier: MPL-2.0+
    SPDX-FileCopyrightText: 2022 Max Chernoff
  ]]

--- Tell the linter about node attributes
--- @class node
--- @field depth integer
--- @field height integer
--- @field id integer
--- @field list node
--- @field next node
--- @field penalty integer
--- @field prev node
--- @field subtype integer

-- Initial setup
lwc = lwc or {}
lwc.name = "lua-widow-control"

-- Locals for `debug_print`
local debug_lib = debug
local string_rep = string.rep
local write_nl = texio.write_nl

local write_log
if status.luatex_engine == "luametatex" then
    write_log = "logfile"
else
    write_log = "log"
```

```
end

--- Prints debugging messages to the log, only if `debug` is set to `true`.
---
--- @param title string The "title" to use
--- @param text string? The "content" to print
--- @return nil
local function debug(title, text)
    if not lwc.debug then return end

    -- The number of spaces we need
    local filler = 15 - #title

    if text then
        write_nl(
            write_log,
            "LWC (" ..
            title ..
            string_rep(" ", filler) ..
            "): " ..
            text
        )
    else
        write_nl(write_log, "LWC: " .. string_rep(" ", 18) .. title)
    end
end

--[[
    \lwc/ is intended to be format-agonistic. It only runs on Lua\TeX{} and
    LuaMeta\TeX{}, but there are still some slight differences between formats.
    Here, we detect the format name then set some flags for later processing.
  ]]
local format = tex.formatname
local context, latex, plain, optex, lmtx

if status.luatex_engine == "luametatex" then
    lmtx = true
end

if format:find("cont") then -- cont-en, cont-fr, cont-nl, ...
    context = true
elseif format:find("latex") then -- lualatex, lualatex-dev, ...
    latex = true
elseif format == "luatex" or
       format == "luahbtex" or
```

```
       format:find("plain")
then -- Plain
    plain = true
elseif format:find("optex") then -- OpTeX
    optex = _G.optex
end

--[[
    Save some local copies of the node library to reduce table lookups.
    This is probably a useless micro-optimization, but it is done in all of the
    ConTeXt and expl3 Lua code, so I should probably do it here too.
  ]]
-- Node ID's
-- (We need to hardcode the subid's sadly)
local id_from_name = node.id
local baselineskip_subid = 2
local glue_id = id_from_name("glue")
local glyph_id = id_from_name("glyph")
local hlist_id = id_from_name("hlist")
local insert_id = id_from_name("insert") or id_from_name("ins")
local line_subid = 1
local linebreakpenalty_subid = 1
local par_id = id_from_name("par") or id_from_name("local_par")
local penalty_id = id_from_name("penalty")
local parfill_subids = {
    parfillleftskip = 17,
    parfillrightskip = 16,
    parinitleftskip = 19,
    parinitrightskip = 18,
}
local vlist_id = id_from_name("vlist")

-- Local versions of globals
local abs = math.abs
local copy = node.copy
local copy_list = node.copy_list or node.copylist
local effective_glue = node.effective_glue or node.effectiveglue
local find_attribute = node.find_attribute or node.findattribute
local free = node.free
local free_list = node.flush_list or node.flushlist
local get_attribute = node.get_attribute or node.getattribute
local hpack = node.hpack
local insert_token = token.put_next or token.putnext
local is_node = node.is_node or node.isnode
```

```
local last = node.slide
local linebreak = tex.linebreak
local new_node = node.new
local remove = node.remove
local set_attribute = node.set_attribute or node.setattribute
local str_byte = string.byte
local str_char = string.char
local str_format = string.format
local subtype = node.subtype
local tex_box = tex.box
local tex_count = tex.count
local tex_dimen = tex.dimen
local tex_lists = tex.lists
local traverse = node.traverse
local traverse_id = node.traverse_id or node.traverseid
local vpack = node.vpack

-- Misc. Constants
local iffalse = token.create("iffalse")
local iftrue = token.create("iftrue")
local INFINITY = 10000
local INSERT_CLASS_MULTIPLE = 1000 * 1000
local INSERT_FIRST_MULTIPLE = 1000
local PAGE_MULTIPLE = 100
local SINGLE_LINE = 50

lwc.colours = {
    expanded = {0.00, 0.70, 0.25},
    failure  = {0.90, 0.00, 0.25},
    moved    = {0.25, 0.25, 1.00},
    cost     = {0.50, 0.50, 0.50},
}

--[[ Package/module initialization.

     Here, we replace any format/engine-specific variables/functions with some
     generic equivalents. This way, we can write the rest of the module without
     worrying about any format/engine differences.
  ]]
local contrib_head,
      draft_offset,
      emergencystretch,
      hold_head,
      info,
```

```
      insert_attribute,
      max_cost,
      page_head,
      paragraph_attribute,
      set_whatsit_field,
      shrink_order,
      stretch_order,
      warning

if lmtx then
    -- LMTX has removed underscores from most of the Lua parts
    debug("LMTX")
    contrib_head = "contributehead"
    shrink_order = "shrinkorder"
    stretch_order = "stretchorder"
    hold_head = "holdhead"
    page_head = "pagehead"
    set_whatsit_field = node.setwhatsitfield
else
    contrib_head = "contrib_head"
    shrink_order = "shrink_order"
    stretch_order = "stretch_order"
    hold_head = "hold_head"
    page_head = "page_head"
    set_whatsit_field = node.setfield
end

if context then
    debug("ConTeXt")

    warning = logs.reporter(lwc.name, "warning")
    local _info = logs.reporter(lwc.name, "info")
    --[[ We don't want the info messages on the terminal, but ConTeXt doesn't
         provide any logfile-only reporters, so we need this hack.
      ]]
    info = function (text)
        logs.pushtarget("logfile")
        _info(text)
        logs.poptarget()
    end
    paragraph_attribute = attributes.public(lwc.name .. "_paragraph")
    insert_attribute = attributes.public(lwc.name .. "_insert")

    -- Dimen/count names
```

```
    emergencystretch = "lwc_emergency_stretch"
    draft_offset = "lwc_draft_offset"
    max_cost = "lwc_max_cost"
elseif plain or latex or optex then
    -- Dimen names
    if tex.isdimen("g__lwc_emergencystretch_dim") then
        emergencystretch = "g__lwc_emergencystretch_dim"
        draft_offset = "g__lwc_draftoffset_dim"
        max_cost = "g__lwc_maxcost_int"
    else
        emergencystretch = "lwcemergencystretch"
        draft_offset = "lwcdraftoffset"
        max_cost = "lwcmaxcost"
    end

    if plain or latex then
        debug("Plain/LaTeX")
        luatexbase.provides_module {
            name = lwc.name,
            date = "2022/11/22", --%%slashdate
            version = "3.0.0", --%%version
            description = [[

This module provides a LuaTeX-based solution to prevent
widows and orphans from appearing in a document. It does
so by increasing or decreasing the lengths of previous
paragraphs.]],
        }
        warning = function(str) luatexbase.module_warning(lwc.name, str) end
        info = function(str) luatexbase.module_info(lwc.name, str) end
        paragraph_attribute = luatexbase.new_attribute(lwc.name .. "_paragraph")
        insert_attribute = luatexbase.new_attribute(lwc.name .. "_insert")
    elseif optex then
        debug("OpTeX")

        warning = function(str) write_nl(lwc.name .. " Warning: " .. str) end
        info = function(str) write_nl("log", lwc.name .. " Info: " .. str) end
        paragraph_attribute = alloc.new_attribute(lwc.name .. "_paragraph")
        insert_attribute = alloc.new_attribute(lwc.name .. "_insert")
    end
else -- This shouldn't ever happen
    error [[Unsupported format.

Please use LaTeX, Plain TeX, ConTeXt or OpTeX.]]
```

```lua
end

-- We can't get the value of \\horigin from Lua, but we can guess it
-- based on the format.
local horigin
if optex or (lmtx and context) then
    horigin = 0
else
    horigin = tex.sp("1in")
end

-- Plain is the only format without a `pre_shipout_filter`
if plain then
    luatexbase.create_callback('pre_shipout_filter', 'list')
end

--[[ Select the fonts

     We want to use cmr6 for the draft mode cost displays, and the easiest
     way to do so is to just hardcode the font id's. This relies on some
     implementation details; however, it is very unlikely to ever be an issue.
  ]]
local SMALL_FONT
if plain then
    SMALL_FONT = 5
elseif latex then
    SMALL_FONT = 6
elseif optex then
    SMALL_FONT = 8
elseif context then
    SMALL_FONT = fonts.definers.define({
        name = "LMRoman6-Regular",
        size = tex.sp("6pt"),
    })
end

-- Global variables
local paragraphs = {} -- The expanded paragraphs on each page
local inserts = {} -- Copies of all the inserts on each page
local costs = {} -- All of the paragraph costs for the document
local pagenum = 1 -- The current page/column number

--[[ Function definitions
  ]]
```

```
--- Gets the current paragraph and page locations
--- @return string
local function get_location()
    return "At " .. pagenum .. "/" .. #paragraphs
end

--- Prints the starting glyphs and glue of an `hlist`.
---
--- Useful for debugging purposes.
---
--- @param head node
--- @return nil
local function get_chars(head)
    if not lwc.debug then return end

    local chars = ""
    for n in traverse(head) do
        if n.id == glyph_id then
            if n.char < 127 then -- Only ASCII
                chars = chars .. str_char(n.char)
            else
                chars = chars .. "#"  -- Replacement for an unknown glyph
            end
        elseif n.id == glue_id then
            chars = chars .. " " -- Any glue goes to a space
        end
        if #chars > 25 then
            break
        end
    end

    debug(get_location(), chars)
end

--- The "cost function" to use. Users can redefine this if they wish.
---
--- @param demerits number The demerits of the broken paragraph
--- @param lines number The number of lines in the broken paragraph
--- @param nat_demerits number The demerits of the naturally-broken paragraph
--- @param nat_lines number The number of lines in the naturally-broken paragraph
--- @param head node The head of the broken paragraph
--- @return number cost The cost of the broken paragraph
function lwc.paragraph_cost(demerits, lines, nat_demerits, nat_lines, head)
    return demerits / math.sqrt(lines)
```

```
end

--- Checks if the ConTeXt "grid snapping" is active
---
--- @return boolean
local function grid_mode_enabled()
    -- Compare the token "mode" to see if `\\ifgridsnapping` is `\\iftrue`
    return token.create("ifgridsnapping").mode == iftrue.mode
end

--- Gets the next node of a specified type/subtype in a node list
---
--- @param head node The head of the node list
--- @param id number The node type
--- @param args table?
---     subtype: number = The node subtype
---     reverse: bool = Whether we should iterate backwards
--- @return node?
local function next_of_type(head, id, args)
    args = args or {}

    if lmtx or not args.reverse then
        for n, subtype in traverse_id(id, head, args.reverse) do
            if (subtype == args.subtype) or (args.subtype == nil) then
                return n
            end
        end
    else
        -- Only LMTX has the built-in backwards traverser, so we need to do
        -- it manually otherwise.
        while head do
            if head.id == id and
               (head.subtype == args.subtype or args.subtype == nil)
            then
                return head
            end
            head = head.prev
        end
    end

    -- Needed for the special `tex.lists` nodes
    if head and head.id == id and
       (head.subtype == args.subtype or args.subtype == nil)
    then
```

```
        return head
    end
end

--- Ensures that a paragraph is ready to be broken
---
--- Only applies to LuaMetaTeX
---
--- @param head node
--- @return nil
local function prepare_linebreak(head)
    if not lmtx then
        return
    end

    -- See how many of the par[left/right][init/fill]skips we have
    local parfills = {}
    local count = 0
    for name, subid in pairs(parfill_subids) do
        parfills[name] = next_of_type(head, glue_id, { subtype = subid })
        if parfills[name] then
            count = count + 1
        end
    end

    if count == 0 then
        -- Usual case
        tex.preparelinebreak(head)
    elseif count == 4 then
        -- Already prepared for some reason, ignored
    else
        -- Uh oh
        warning("Weird par(fill/init)skips found!")
        tex.preparelinebreak(head) -- Try to fix it
    end
end

--- Breaks a paragraph one line longer than natural
---
--- @param head node The unbroken paragraph
--- @param parfillskip table<number> The {width, stretch, shrink,
---                                  stretch_order, shrink_order} to set
---                                  for the \\parfillskip
--- @return node long_node The broken paragraph
```

```
--- @return table long_info An info table about the broken paragraph
local function long_paragraph(head, parfillskip)
    -- We can't modify the original paragraph
    head = copy_list(head)

    prepare_linebreak(head)

    -- TODO node.setglue is broken in LMTX, so we have to do this manually
    local n = last(head)
    n.width = parfillskip[1]
    n.stretch = parfillskip[2]
    n.shrink = parfillskip[3]
    n[stretch_order] = parfillskip[4]
    n[shrink_order] = parfillskip[5]

    -- Break the paragraph 1 line longer than natural
    local long_node, long_info =  linebreak(head, {
        looseness = 1,
        emergencystretch = tex_dimen[emergencystretch],
    })

    -- Mark the last line for the costs display
    set_attribute(
        last(long_node),
        paragraph_attribute,
        -1 * (#paragraphs + 1 + (PAGE_MULTIPLE * pagenum))
    )

    return long_node, long_info
end

--- Breaks a paragraph at its natural length
---
--- @param head node The unbroken paragraph
--- @return table natural_info An info table about the broken paragraph
local function natural_paragraph(head)
    -- We can't modify the original paragraph
    head = copy_list(head)

    prepare_linebreak(head)

    -- Break the paragraph naturally to get \\prevgraf
    local natural_node, natural_info = linebreak(head)
    free_list(natural_node)

    return natural_info
```

```
end

local show_colours = false
--- Changes the text colour in a node list if draft mode is active
---
--- @param head node The first node to colour
--- @param colour string The name of a colour in `lwc.colours`
--- @return node head The coloured node
local function colour_list(head, colour)
    if not show_colours then
        return head
    end

    local pdf_colour = str_format(
        "%.2f %.2f %.2f rg",
        table.unpack(lwc.colours[colour])
    )

    if optex and optex.set_node_color then
        for n in node.traverse(head) do
            optex.set_node_color(n, pdf_colour)
        end

        return head
    end

    if context then
        nodes.tracers.colors.setlist(head, "lwc_" .. colour)
        return head
    end

    -- Adapted from https://tex.stackexchange.com/a/372437 and
    -- https://github.com/zauguin/luametalatex/issues/8.
    local start_colour = new_node("whatsit", subtype("pdf_colorstack"))
    set_whatsit_field(start_colour, "stack", 0)
    set_whatsit_field(start_colour, "command", 1)
    set_whatsit_field(start_colour, "data", pdf_colour)

    local end_colour = new_node("whatsit", subtype("pdf_colorstack"))
    set_whatsit_field(end_colour, "stack", 0)
    set_whatsit_field(end_colour, "command", 2)

    start_colour.next = head
    last(head).next = end_colour

    return start_colour
```

```
end

--- Saves each paragraph, but lengthened by 1 line
---
--- Called by the `pre_linebreak_filter` callback
---
--- @param head node The pre-broken paragraph
--- @return node head The unmodified `head` argument
function lwc.save_paragraphs(head)
    if (head.id ~= par_id and context) or -- Make sure that `head` is a paragraph
        status.output_active or -- Don't run during the output routine
        tex.nest.ptr > 1 -- Don't run inside boxes
    then
        return head
    end

    -- Prevent the "underfull hbox" warnings when we store a potential paragraph
    local renable_box_warnings
    if (context or optex) or
       #luatexbase.callback_descriptions("hpack_quality") == 0
    then -- See #18 and michal-h21/linebreaker#3
        renable_box_warnings = true
        lwc.callbacks.disable_box_warnings.enable()
    end

    natural_info = natural_paragraph(head)

    -- Prevent ultra-short last lines (\TeX{}book p. 104). Equivalent to
    -- \\parfillskip=0.75\\hsize plus 0.05\\hsize minus 0.75\\hsize.
    -- From http://petr.olsak.net/ftp/olsak/tbn/tbn.pdf p. 234 (via Jan Sustek)
    long_node, long_info = long_paragraph(
        head,
        {0.75 * tex.hsize, 0.05 * tex.hsize, 0.75 * tex.hsize, 0, 0}
    )

    if long_info.prevgraf ~= natural_info.prevgraf + 1 then
        -- The \\parfillskip settings with \\looseness=1 can sometimes
        -- lengthen paragraphs by two lines instead of one. If this happens,
        -- we fall back to a slightly-worse \\parfillskip setting.
        free_list(long_node)
        long_node, long_info = long_paragraph(
            head,
            {0, 0.8 * tex.hsize, false, 0, false}
        )
```

```
end

if renable_box_warnings then
    lwc.callbacks.disable_box_warnings.disable()
end

if not grid_mode_enabled() then
    -- Offset the \\prevdepth differences between natural and long
    local prevdepth = new_node("glue")
    prevdepth.width = natural_info.prevdepth - long_info.prevdepth
    last(long_node).next = prevdepth
end

local long_cost = lwc.paragraph_cost(
    long_info.demerits,
    long_info.prevgraf,
    natural_info.demerits,
    natural_info.prevgraf,
    long_node
)

if long_info.prevgraf ~= natural_info.prevgraf + 1 or
   long_cost < 10 -- Any paragraph that is "free" to expand is suspicious
then
    -- This paragraph is infinitely bad
    long_cost = math.maxinteger
end

-- The initial glue can disappear in ConTeXt's grid mode, so we
-- save starting at the first hlsit
local saved_node = next_of_type(long_node, hlist_id, { subtype = line_subid })

for n in traverse_id(hlist_id, saved_node) do
    n.list = colour_list(n.list, "expanded")
end

table.insert(paragraphs, {
    cost = long_cost,
    node = copy_list(saved_node)
})

free_list(long_node)

costs[#paragraphs + (PAGE_MULTIPLE * pagenum)] = long_cost

-- Print some debugging information
```

```
    if lwc.debug then
        get_chars(head)
        debug(get_location(), "nat  lines    " .. natural_info.prevgraf)
        debug(
            get_location(),
            "nat  cost " ..
            lwc.paragraph_cost(natural_info.demerits, natural_info.prevgraf)
        )
        debug(get_location(), "long lines    " .. long_info.prevgraf)
        debug(
            get_location(),
            "long cost " ..
            lwc.paragraph_cost(long_info.demerits, long_info.prevgraf)
        )
    end

    -- \ConTeXt{} crashes if we return `true`
    return head
end

--- Tags the beginning and the end of each paragraph as it is added to the page.
---
--- We add an attribute to the first and last node of each paragraph. The ID is
--- some arbitrary number for \lwc/, and the value corresponds to the
--- paragraphs index, which is negated for the end of the paragraph.
---
--- @param head node
--- @return nil
local function mark_paragraphs(head)
    -- Tag the paragraphs
    if status.output_active then
        -- Don't run during the output routine
        return
    end

    -- Get the start and end of the paragraph
    local top = next_of_type(head, hlist_id, { subtype = line_subid })
    local bottom = last(head)

    -- The inserts disappear before `pre_output_routine`, so we shouldn't
    -- mark them.
    while bottom.id == insert_id do
        bottom = bottom.prev
    end
```

```
    if top ~= bottom then
        set_attribute(
            top,
            paragraph_attribute,
            #paragraphs + (PAGE_MULTIPLE * pagenum)
        )
        set_attribute(
            bottom,
            paragraph_attribute,
            -1 * (#paragraphs + (PAGE_MULTIPLE * pagenum))
        )
    else
        -- We need a special tag for a 1-line paragraph since the node can only
        -- have a single attribute value
        set_attribute(
            top,
            paragraph_attribute,
            #paragraphs + (PAGE_MULTIPLE * pagenum) + SINGLE_LINE
        )
    end
end

--- Tags the each line with the indices of any corresponding inserts.
---
--- We need to tag the first element of the hlist before the any insert nodes
--- since the insert nodes are removed before `pre_output_filter` gets called.
---
--- @param head node
--- @return nil
local function mark_inserts(head)
    local insert_indices = {}
    for insert in traverse_id(insert_id, head) do
        -- Save the found insert nodes for later
        inserts[#inserts+1] = copy(insert)

        -- Tag the insert's content so that we can find it later
        set_attribute(insert.list, insert_attribute, #inserts)

        -- We need to tag all lines---not just the start and the end---since
        -- \TeX{} can split the insert between pages at any point.
        for n in traverse(insert.list.next) do
            set_attribute(n, insert_attribute, -1 * #inserts)
        end
```

```
        -- Each hlist/line can have multiple inserts, but so we can't just tag
        -- the hlist as we go. Instead, we need save up all of their indices,
        -- then tag the hlist with the first and last indices.
        insert_indices[#insert_indices+1] = #inserts

        if not insert.next or
           insert.next.id ~= insert_id
        then
            local hlist_before = next_of_type(insert, hlist_id, { reverse = true} )

            local insert_class
            if lmtx then
                insert_class = insert.index
            else
                insert_class = insert.subtype
            end

            -- We tag the first element of the hlist/line with an integer
            -- that holds the insert class and the first and last indices
            -- of the inserts contained in the line. This won't work if
            -- the line has multiple classes of inserts, but I don't think
            -- that happens in real-world documents. If this does turn out
            -- to be an issue, we can get the insert's class from it's copy
            -- at `pre_output_filter` instead of saving it now.
            set_attribute(
                hlist_before.list,
                insert_attribute,
                insert_class      * INSERT_CLASS_MULTIPLE +
                insert_indices[1] * INSERT_FIRST_MULTIPLE +
                insert_indices[#insert_indices]
            )

            -- Clear the indices to prepare for the next line
            insert_indices = {}
        end
    end
end

--- Saves the inserts and tags a typeset paragraph. Called by the
--- `post_linebreak_filter` callback.
---
--- @param head node The head of the broken paragraph
--- @return node head The unmodified `head` parameter
function lwc.mark_paragraphs(head)
```

```
    mark_paragraphs(head)
    mark_inserts(head)

    return head
end

--- Checks to see if a penalty matches the widow/orphan/broken penalties
---
--- @param penalty number
--- @return boolean
local function is_matching_penalty(penalty)
    local widowpenalty = tex.widowpenalty
    local clubpenalty = tex.clubpenalty
    local displaywidowpenalty = tex.displaywidowpenalty
    local brokenpenalty = tex.brokenpenalty

    penalty = penalty - tex.interlinepenalty

    -- Adapted from https://tug.org/TUGboat/tb39-3/tb123mitt-widows-code.pdf.
    -- This only takes into account the original \TeX{} penalties, not the
    -- "new" \eTeX{} \\(club/widow/broken)penalties commands.
    return penalty ~= 0 and
           penalty <  INFINITY and (
               penalty == widowpenalty or
               penalty == displaywidowpenalty or
               penalty == clubpenalty or
               penalty == clubpenalty + widowpenalty or
               penalty == clubpenalty + displaywidowpenalty or
               penalty == brokenpenalty or
               penalty == brokenpenalty + widowpenalty or
               penalty == brokenpenalty + displaywidowpenalty or
               penalty == brokenpenalty + clubpenalty or
               penalty == brokenpenalty + clubpenalty + widowpenalty or
               penalty == brokenpenalty + clubpenalty + displaywidowpenalty
           )
end

--- Determines if we should "activate" \lwc/ for the current page/column.
---
--- Users can redefine this if they wish.
---
--- @param penalty number The \\outputpenalty for the current page/column
--- @param paragraphs table<table<string, node|number>> The `paragraphs` table
--- @param head node The head of the current page/column
```

```
--- @return boolean activate True if \lwc/ should move the last line on this page
function lwc.should_remove_widows(penalty, paragraphs, head)
    return is_matching_penalty(penalty)
end

--- Reset any state saved between pages
---
--- This function is *vital* to ensure that we don't leak any nodes.
--- If we do leak nodes, then very large documents will slow down and
--- eventually fail to compile.
---
--- @return nil
local function reset_state()
    for _, paragraph in ipairs(paragraphs) do
        free_list(paragraph.node)
    end
    paragraphs = {}

    for _, insert in ipairs(inserts) do
        free(insert)
    end
    inserts = {}

    pagenum = pagenum + 1
end

--- When we are unable to remove a widow/orphan, print a warning
---
--- @return nil
local function remove_widows_fail()
    warning("Widow/Orphan/Broken Hyphen NOT removed on page " .. pagenum)

    local last_line = next_of_type(
        last(tex_lists[page_head]),
        hlist_id,
        { subtype = line_subid, reverse = true }
    )
    if last_line then
        last_line.list = colour_list(last_line.list, "failure")
    end

    local next_first_line = next_of_type(
        tex_lists[contrib_head],
        hlist_id,
        { subtype = line_subid }
```

```
    )
    if next_first_line then
        next_first_line.list = colour_list(next_first_line.list, "failure")
    end

    reset_state()
end

--- Finds the first and last paragraphs present on a page
---
--- @param head node The node representing the start of the page
--- @return number first_index The index of the first paragraph on the page in
---                            the `paragraphs` table
--- @return number last_index The index of the last paragraph on the page in the
---                           `paragraphs` table
local function first_last_paragraphs(head)
    local first_index, last_index

    -- Find the last paragraph on the page, starting at the end, heading in reverse
    local n = last(head)
    while n do
        local value = get_attribute(n, paragraph_attribute)
        if value then
            last_index = value % PAGE_MULTIPLE
            break
        end

        n = n.prev
    end

    -- Find the first paragraph on the page, from the top
    local first_val, first_head = find_attribute(head, paragraph_attribute)
    while abs(first_val) // PAGE_MULTIPLE == pagenum - 1 do
        -- If the first complete paragraph on the page was initially broken on the
        -- previous page, then we can't expand it here. Why can't we expand it?
        -- Well, expanding it will nearly always change how the first few lines
        -- are printed, but we can't modify those since they've already been
        -- shipped out. So, we need to skip these paragraphs.
        first_val, first_head = find_attribute(
            first_head.next,
            paragraph_attribute
        )
    end

    first_index = first_val % PAGE_MULTIPLE
```

```
    if first_index >= SINGLE_LINE then
        first_index = first_index - SINGLE_LINE
    end

    debug("first/last", first_index .. "/" .. last_index)

    return first_index, last_index
end

--- Selects the "best" paragraph on the page to expand
---
--- @param head node The node representing the start of the page
--- @return number? best_index The index of the paragraph to expand in the
---                            `paragraphs` table
local function best_paragraph(head)
    local first_paragraph_index, last_paragraph_index = first_last_paragraphs(head)

    -- Find the paragraph on the page with the least cost.
    local best_index = 1
    local best_cost = paragraphs[best_index].cost

    -- We find the current "best" replacement
    for index, paragraph in pairs(paragraphs) do
        if paragraph.cost < best_cost and
           index <  last_paragraph_index and
           index >= first_paragraph_index
        then
            best_index, best_cost = index, paragraph.cost
        end
    end

    debug(
        "selected para",
        pagenum .. "/" .. best_index .. " (" .. best_cost .. ")"
    )

    if best_cost  >  tex_count[max_cost] or
       best_index == last_paragraph_index or -- Shouldn't happen
       best_cost  == math.maxinteger
    then
        return nil
    else
        return best_index
    end
end
```

```
--- Gets any inserts present in the moved line
---
--- @param last_line node The moved last line
--- @return table<node> inserts A list of the present inserts
local function get_inserts(last_line)
    local selected_inserts = {}

    local n = last_line.list
    while n do -- Iterate through the last line
        local line_value
        line_value, n = find_attribute(n, insert_attribute)

        if not n then
            break
        end

        -- Demux the insert values
        local class = line_value // INSERT_CLASS_MULTIPLE
        local first_index = (line_value % INSERT_CLASS_MULTIPLE)
                            // INSERT_FIRST_MULTIPLE
        local last_index = line_value % INSERT_FIRST_MULTIPLE

        -- Get the output box containing the insert boxes
        local insert_box

        if lmtx then
            insert_box = tex.getinsertcontent(class)
            -- `getinsertcontent` resets the insert box, so we need to
            -- re-set it.
            tex.setinsertcontent(class, insert_box)
        else
            insert_box = tex_box[class]
        end

        -- Get any portions of the insert "held over" until the next page
        local split_insert
        if lmtx then
            split_insert = next_of_type(
                tex_lists[hold_head],
                insert_id,
                { index = class }
            )
        else
            split_insert = next_of_type(
                tex_lists[hold_head],
```

```
        insert_id,
        { subtype = class }
    )
end

-- We use the same procedure for this page and the next page
for i, insert in ipairs { insert_box, split_insert } do
    local m = insert and insert.list

    while m do -- Iterate through the insert box
        local box_value
        box_value, m = find_attribute(m, insert_attribute)
        local next = m.next

        if not m then
            break
        end

        if abs(box_value) >= first_index and
           abs(box_value) <= last_index
        then
            -- Remove the respective contents from the insert box
            insert.list = remove(insert.list, m)

            if box_value > 0 and i == 1 then
                table.insert(selected_inserts, copy(inserts[box_value]))
            end

            free(m)
        end

        m = next
    end
end

-- If the box has no contents, then void it so that any \\ifvoid
-- tests work correctly in the output routine.
if not insert_box.list then
    tex_box[class] = nil
end

if split_insert and not split_insert.list then
    remove(tex_lists[hold_head], split_insert)
end

n = n.next
```

```
    end

    if #selected_inserts ~= 0 then
        -- This is a somewhat-risky process, so we print an info
        -- message just in case something goes wrong. We can probably
        -- remove this in the future if we're sure that everything
        -- is working correctly.
        info("Moving footnotes on page " .. pagenum)
    end

    return selected_inserts
end

lwc.nobreak_behaviour = "keep"
--- Moves the last line of the page onto the following page.
---
--- This is the most complicated function of the module since it needs to
--- look back to see if there is a heading preceding the last line, then it does
--- some low-level node shuffling.
---
--- @param head node The node representing the start of the page
--- @return boolean success Set to false if something went wrong
local function move_last_line(head)
    -- Start of final paragraph
    debug("remove_widows", "moving last line")

    -- Here we check to see if the widow/orphan was preceded by a large penalty
    local big_penalty_found, last_line, hlist_head
    local n = last(head).prev
    while n do
        if n.id == glue_id then
            -- Ignore any glue nodes
        elseif n.id == penalty_id and n.penalty >= INFINITY then
            -- Infinite break penalty
            big_penalty_found = true
        elseif big_penalty_found and n.id == hlist_id then
            -- Line before the penalty
            if lwc.nobreak_behaviour == "keep" then
                hlist_head = n
                big_penalty_found = false
            elseif lwc.nobreak_behaviour == "split" then
                n = last(head)
                break
            elseif lwc.nobreak_behaviour == "warn" then
```

```
            debug("last line", "heading found")
            return false
        end
    else
        -- Not found
        if hlist_head then
            n = hlist_head
        else
            n = last(head)
        end
        break
    end
    n = n.prev
end

local potential_penalty = n.prev.prev

if potential_penalty and
   potential_penalty.id      == penalty_id and
   potential_penalty.subtype == linebreakpenalty_subid and
   is_matching_penalty(potential_penalty.penalty)
then
    warning("Making a new widow/orphan/broken hyphen on page " .. pagenum)

    local second_last_line = next_of_type(
        potential_penalty,
        hlist_id,
        { subtype = line_subid, reverse = true }
    )
    second_last_line.list = colour_list(second_last_line.list, "failure")
end

last_line = copy_list(n)

last_line.list = colour_list(last_line.list, "moved")

-- Reinsert any inserts originally present in this moved line
local selected_inserts = get_inserts(last_line)
for _, insert in ipairs(selected_inserts) do
    last(last_line).next = insert
end

-- Add back in the content from the next page
last(last_line).next = copy_list(tex_lists[contrib_head])
```

```
    free_list(n.prev.prev.next)
    n.prev.prev.next = nil

    -- Set the content of the next page
    free_list(tex_lists[contrib_head])
    tex_lists[contrib_head] = last_line

    return true
end

--- Replace the chosen paragraph with its expanded version.
---
--- This is the "core function" of the module since it is what ultimately causes
--- the expansion to occur.
---
--- @param head node
--- @param paragraph_index number
local function replace_paragraph(head, paragraph_index)
    -- Remove any inserts. They are completely ignored after
    -- the page has been broken, but they can upset LuaMetaLaTeX if
    -- they're found somewhere unexpected.
    local target_node, last_target_node
    for n in traverse(paragraphs[paragraph_index].node) do
        -- Just removing the inserts from the list doesn't work properly,
        -- so we instead copy over everything that isn't an insert.
        if n.id ~= insert_id then
            if not target_node then
                target_node = copy(n)
                last_target_node = target_node
            else
                last_target_node.next = copy(n)
                last_target_node = last_target_node.next
            end
        end
    end

    local start_found = false
    local end_found = false
    local free_nodes_begin

    -- Loop through all of the nodes on the page with the \lwc/ attribute
    local n = head
    while n do
        local value
```

```
value, n = find_attribute(n, paragraph_attribute)

if not n then
    break
end

debug("remove_widows", "found " .. value)

-- Insert the start of the replacement paragraph
if value == paragraph_index + (PAGE_MULTIPLE * pagenum) or
   value == paragraph_index + (PAGE_MULTIPLE * pagenum) + SINGLE_LINE
then
    debug("remove_widows", "replacement start")
    start_found = true

    -- Fix the `\\baselineskip` glue between paragraphs
    height_difference = (
        next_of_type(n, hlist_id, { subtype = line_subid }).height -
        next_of_type(
            target_node, hlist_id, { subtype = line_subid }
        ).height
    )

    local prev_bls = next_of_type(
        n,
        glue_id,
        { subtype = baselineskip_subid, reverse = true }
    )

    if prev_bls then
        prev_bls.width = prev_bls.width + height_difference
    end

    n.prev.next = target_node
    free_nodes_begin = n
end

-- Insert the end of the replacement paragraph
if value == -(paragraph_index + (PAGE_MULTIPLE * pagenum)) or
   value ==   paragraph_index + (PAGE_MULTIPLE * pagenum) + SINGLE_LINE
then
    debug("remove_widows", "replacement end")
    end_found = true

    local target_node_last = last(target_node)
```

```
            if grid_mode_enabled() then
                -- Account for the difference in depth
                local after_glue = new_node("glue")
                after_glue.width = n.depth - target_node_last.depth
                target_node_last.next = after_glue

                after_glue.next = n.next
            else
                target_node_last.next = n.next
            end

            n.next = nil

            break
        end

        n = n.next
    end

    if start_found and end_found then
        free_list(free_nodes_begin)
    else
        warning("Paragraph NOT expanded on page " .. pagenum)
    end
end

--- Remove the widows and orphans from the page, just after the output routine.
---
--- This is called just after the end of the output routine, before the page is
--- shipped out. If the output penalty indicates that the page was broken at a
--- widow or an orphan, we replace one paragraph with the same paragraph, but
--- lengthened by one line. Then, we can push the bottom line of the page to the
--- next page.
---
--- @param head node
--- @return node
function lwc.remove_widows(head)
    debug("outputpenalty", tex.outputpenalty .. " " .. #paragraphs)

    -- See if there is a widow/orphan for us to remove
    if not lwc.should_remove_widows(tex.outputpenalty, paragraphs, head) then
        reset_state()
        return head
    end
```

```
info("Widow/orphan/broken hyphen detected. Attempting to remove")

-- Nothing that we can do if there aren't any paragraphs available to expand
if #paragraphs == 0 then
    debug("failure", "no paragraphs to expand")
    remove_widows_fail()
    return head
end

-- Check the original height of \\box255
local vsize = tex_dimen.vsize
local orig_vpack = vpack(head)
local orig_height_diff = orig_vpack.height - vsize
orig_vpack.list = nil
free(orig_vpack)

-- Find the paragraph to expand
local paragraph_index = best_paragraph(head)

if not paragraph_index then
    debug("failure", "no good paragraph")
    remove_widows_fail()
    return head
end

-- Move the last line of the page to the next page
if not move_last_line(head) then
    debug("failure", "can't move last line")
    remove_widows_fail()
    return head
end

-- Replace the chosen paragraph with its expanded version
replace_paragraph(head, paragraph_index)

-- The final \\box255 needs to be exactly \\vsize tall to avoid
-- over/underfull box warnings, so we correct any discrepancies
-- here.
local new_vpack = vpack(head)
local new_height_diff = new_vpack.height - vsize
new_vpack.list = nil
free(new_vpack)
-- We need the original height discrepancy in case there are \\vfill's
local net_height_diff = orig_height_diff - new_height_diff
local bls = tex.skip.baselineskip
```

```
    local bls_width = bls.width
    free(bls)

    if abs(net_height_diff) > 0 and
       -- A difference larger than 0.25\\baselineskip is probably not from
       -- \lwc/, so we let those warnings surface
       abs(net_height_diff) < bls_width / 4
    then
        local bottom_glue = new_node("glue")
        bottom_glue.width = net_height_diff
        last(head).next = bottom_glue
    end

    info(
        "Widow/orphan/broken hyphen successfully removed at paragraph "
        .. paragraph_index
        .. " on page "
        .. pagenum
    )

    reset_state()

    return head
end

local show_costs = false
--- Add the paragraph to the list of paragraphs on the page.
---
--- Called immediately before the page is shipped out so that we can get
--- the costs on the correct side in multi-column layouts.
---
--- To evenly align all of the costs in the margins, we need to know the
--- the exact position of the start and end of the paragraph on the page.
--- This is surprisingly complicated.
---
--- @param head node The box to be shipped out
--- @return true
function lwc.show_costs (head)
    if not show_costs then
        return true
    end

    local pagewidth = tex.pagewidth or layouts.getpagedimensions()

    --- Loop over each sublist, add up the total width, and show the costs.
```

```
---
--- @param n node The node to loop over
--- @param width number The accumulated width so far
--- @param parent node The parent node of the current list
--- @return nil
local function recurse(n, width, parent)
    for m in traverse(n) do
        -- Anything with an \\hbox parent and a width is actual width.
        -- (If it had a \\vbox parent, then n.width would actually be height.)
        local self_width = 0
        if m.id == glue_id and parent.id == hlist_id then
            self_width = effective_glue(m, parent)
        elseif m.width and parent.id == hlist_id then
            self_width = m.width
        end

        -- A node's "shift" attribute is horizontal only if the parent is
        -- a \\vbox. This corresponds to the primatives \\moveleft and
        -- \\moveright. (If the parent is a \\hbox, then n.shift is
        -- vertical and corresponds to \\raise.)
        local shift = 0
        if m.shift and
           (parent.id == vlist_id or
            not is_node(parent))
        then
            shift = m.shift
        end

        width = width + self_width

        local attr = get_attribute(m, paragraph_attribute)
        if attr and abs(attr) % PAGE_MULTIPLE >= SINGLE_LINE then
            attr = -1 * (abs(attr) - SINGLE_LINE)
        end

        local cost = costs[abs(attr or 0)]

        if attr and attr < 0 and cost and m.list then
            -- We've found the end of a marked paragraph!

            -- Generate the \\hbox containing the formatted cost
            local cost_str
            if not cost then
                return
            elseif cost < math.maxinteger then
```

```
    cost_str = str_format("%.0f", cost)
else
    cost_str = "infinite"
end

local prev, first
for letter in cost_str:gmatch(".")  do
    local curr = new_node("glyph")
    curr.font = SMALL_FONT
    curr.char = str_byte(letter)

    if not first then
        first = curr
    else
        prev.next = curr
    end
    prev = curr
end

local text = hpack(colour_list(first, "cost"))

-- Make an \\hss to make sure that our `\\hbox`es aren't overfull
local hss = new_node("glue")
hss.stretch = 1
hss[stretch_order] = 1
hss.shrink = 1
hss[shrink_order] = 1

local hbox
local offset = new_node("glue")

if (width >= pagewidth / 2) or
   (m.width >= 0.4 * pagewidth)
then -- Right column or single-column
    -- Costs in the right margin
    offset.width = (
        pagewidth -
        width -
        m.width -
        shift -
        tex_dimen[draft_offset]
    )
    text.next = hss
    hbox = hpack(text, 0, "exactly")
else -- Left column
```

```
                    -- Costs in the left margin
                    offset.width = (
                        tex_dimen[draft_offset] -
                        m.width -
                        width -
                        shift
                    )
                    hss.next = text
                    hbox = hpack(hss, 0, "exactly")
                end

                last(m.list).next = offset
                offset.next = hbox
            elseif m.list then
                recurse(m.list, width - self_width + shift, m)
            end
        end
    end

    -- Start at the root of the page
    recurse(head.list, (tex.hoffset or 0) + horigin, {})

    -- LaTeX requires us to always return true here
    return true
end

--- Create a table of functions to enable or disable a given callback
---
--- @param t table Parameters of the callback to create
---     callback: string = The \LuaTeX{} callback name
---     func: function = The function to call
---     name: string = The name/ID of the callback
---     category: string = The category for a \ConTeXt{} "Action"
---     position: string = The "position" for a \ConTeXt{} "Action"
---     lowlevel: boolean = If we should use a lowlevel \LuaTeX{} callback
---                         instead of a \ConTeXt{} "Action"
--- @return table t Enablers/Disablers for the callback
---     enable: function = Enable the callback
---     disable: function = Disable the callback
local function register_callback(t)
    if plain or latex then -- Both use \LuaTeX{}Base for callbacks
        return {
            enable = function()
                luatexbase.add_to_callback(t.callback, t.func, t.name)
```

```
            end,
            disable = function()
                luatexbase.remove_from_callback(t.callback, t.name)
            end,
        }
    elseif context and not t.lowlevel then
        return {
            -- Register the callback when the table is created,
            -- but activate it when `enable()` is called.
            enable = nodes.tasks.appendaction(t.category, t.position, "lwc." .. t.name)
                or function()
                    nodes.tasks.enableaction(t.category, "lwc." .. t.name)
                end,
            disable = function()
                nodes.tasks.disableaction(t.category, "lwc." .. t.name)
            end,
        }
    elseif context and t.lowlevel then
        -- Some of the callbacks in \ConTeXt{} have no associated "actions".
        -- Unlike with \LuaTeX{}base, \ConTeXt{} leaves some \LuaTeX{} callbacks
        -- unregistered and unfrozen. Because of this, we need to register some
        -- callbacks at the engine level. This is fragile though, because a
        -- future \ConTeXt{} update may decide to register one of these
        -- functions, in which case \lwc/ will crash with a cryptic error
        -- message.
        return {
            enable = function() callback.register(t.callback, t.func) end,
            disable = function() callback.register(t.callback, nil) end,
        }
    elseif optex then -- Op\TeX{} is luckily very similar to luatexbase
        return {
            enable = function()
                callback.add_to_callback(t.callback, t.func, t.name)
            end,
            disable = function()
                callback.remove_from_callback(t.callback, t.name)
            end,
        }
    end
end

-- Add all of the callbacks
lwc.callbacks = {
```

```
    disable_box_warnings = register_callback({
        callback = "hpack_quality",
        func     = function() end,
        name     = "disable_box_warnings",
        lowlevel = true,
    }),
    remove_widows = register_callback({
        callback = "pre_output_filter",
        func     = lwc.remove_widows,
        name     = "remove_widows",
        lowlevel = true,
    }),
    save_paragraphs = register_callback({
        callback = "pre_linebreak_filter",
        func     = lwc.save_paragraphs,
        name     = "save_paragraphs",
        category = "processors",
        position = "after",
    }),
    mark_paragraphs = register_callback({
        callback = "post_linebreak_filter",
        func     = lwc.mark_paragraphs,
        name     = "mark_paragraphs",
        category = "finalizers",
        position = "after",
    }),
    show_costs = register_callback({
        callback = "pre_shipout_filter",
        func     = lwc.show_costs,
        name     = "show_costs",
        category = "shipouts",
        position = "finishers",
    }),
}

local lwc_enabled = false
--- Enables the paragraph callbacks
function lwc.enable_callbacks()
    debug("callbacks", "enabling")
    if not lwc_enabled then
        lwc.callbacks.save_paragraphs.enable()
        lwc.callbacks.mark_paragraphs.enable()

        lwc_enabled = true
```

```
    else
        info("Already enabled")
    end
end

--- Disables the paragraph callbacks
function lwc.disable_callbacks()
    debug("callbacks", "disabling")
    if lwc_enabled then
        lwc.callbacks.save_paragraphs.disable()
        lwc.callbacks.mark_paragraphs.disable()

        -- We do *not* disable `remove_widows` callback, since we still want
        -- to expand any of the previously-saved paragraphs if we hit an orphan
        -- or a widow.
        lwc_enabled = false
    else
        info("Already disabled")
    end
end

function lwc.if_lwc_enabled()
    debug("iflwc")
    if lwc_enabled then
        insert_token(iftrue)
    else
        insert_token(iffalse)
    end
end

--- Mangles a macro name so that it's suitable for a specific format
---
--- @param name string The plain name
--- @param args table<string> The TeX types of the function arguments
--- @return string name The mangled name
local function mangle_name(name, args)
    if plain then
        return "lwc@" .. name:gsub("_", "@")
    elseif optex then
        return "_lwc_" .. name
    elseif context then
        return "lwc_" .. name
    elseif latex then
        return "__lwc_" .. name .. ":" .. string_rep("n", #args)
```

```
    end
end

--- Creates a TeX command that evaluates a Lua function
---
--- @param name string The name of the csname to define
--- @param func function
--- @param args table<string> The TeX types of the function arguments
--- @return nil
local function register_tex_cmd(name, func, args)
    local scanning_func
    name = mangle_name(name, args)

    if not context then
        local scanners = {}
        for _, arg in ipairs(args) do
            scanners[#scanners+1] = token['scan_' .. arg]
        end

        -- An intermediate function that properly "scans" for its arguments
        -- in the \TeX{} side.
        scanning_func = function()
            local values = {}
            for _, scanner in ipairs(scanners) do
                values[#values+1] = scanner()
            end

            func(table.unpack(values))
        end
    end

    if optex then
        define_lua_command(name, scanning_func)
        return
    elseif plain or latex then
        local index = luatexbase.new_luafunction(name)
        lua.get_functions_table()[index] = scanning_func
        token.set_lua(name, index)
    elseif context then
        interfaces.implement {
            name = name,
            public = true,
            arguments = args,
            actions = func
```

```
        }
    end
end

--[[ Make all of the \lwc/ Lua commands available from \TeX{}
  ]]
register_tex_cmd("if_enabled", lwc.if_lwc_enabled, {})
register_tex_cmd("enable", lwc.enable_callbacks, {})
register_tex_cmd("disable", lwc.disable_callbacks, {})
register_tex_cmd(
    "nobreak",
    function(str)
        lwc.nobreak_behaviour = str
    end,
    { "string" }
)
register_tex_cmd(
    "debug",
    function(str)
        lwc.debug = str ~= "0" and str ~= "false" and str ~= "stop"
    end,
    { "string" }
)
register_tex_cmd(
    "show_costs",
    function(str)
        show_costs = str ~= "0" and str ~= "false" and str ~= "stop"
    end,
    { "string" }
)
register_tex_cmd(
    "show_colours",
    function(str)
        show_colours = str ~= "0" and str ~= "false" and str ~= "stop"
    end,
    { "string" }
)
register_tex_cmd(
    "pre_shipout",
    function(box)
        luatexbase.call_callback('pre_shipout_filter', tex_box[box])
    end,
    { "int" }
```

```
)

--- Silence the luatexbase "Enabling/Removing <callback>" info messages
---
--- Every time that a paragraph is typeset, \lwc/ hooks in
--- and typesets the paragraph 1 line longer. Some of these longer paragraphs
--- will have pretty bad badness values, so TeX will issue an over/underfull
--- hbox warning. To block these warnings, we hook into the `hpack_quality`
--- callback and disable it so that no warning is generated.
---
--- However, each time that we enable/disable the null `hpack_quality` callback,
--- luatexbase puts an info message in the log. This completely fills the log file
--- with useless error messages, so we disable it here.
---
--- This uses the Lua `debug` library to internally modify the log upvalue in the
--- `add_to_callback` function. This is almost certainly a terrible idea, but I
--- don't know of a better way to do it.
---
--- @return nil
local function silence_luatexbase()
    local nups = debug_lib.getinfo(luatexbase.add_to_callback).nups

    for i = 1, nups do
        local name, func = debug_lib.getupvalue(luatexbase.add_to_callback, i)
        if name == "luatexbase_log" then
            debug_lib.setupvalue(
                luatexbase.add_to_callback,
                i,
                function(text)
                    if text:match("^Inserting") or text:match("^Removing") then
                        return
                    else
                        func(text)
                    end
                end
            )
            return
        end
    end
end

-- Call `silence_luatexbase` in Plain and LaTeX, unless the undocmented global
-- `LWC_NO_DEBUG` is set. We provide this opt-out in case something goes awry
-- with the `debug` library calls.
```

```lua
if (plain or latex) and
   not LWC_NO_DEBUG --- @diagnostic disable-line
then
    silence_luatexbase()
end

-- Register colours for ConTeXt
if context then
    for colour, values in pairs(lwc.colours) do
        attributes.colors.defineprocesscolor(
            "lwc_" .. colour,
            str_format("r=%.2f, g=%.2f, b=%.2f", table.unpack(values))
        )
    end
end

-- Activate \lwc/
lwc.callbacks.remove_widows.enable()
lwc.callbacks.show_costs.enable()

return lwc
```

lua-widow-control.tex

```
% lua-widow-control
% https://github.com/gucci-on-fleek/lua-widow-control
% SPDX-License-Identifier: MPL-2.0+
% SPDX-FileCopyrightText: 2022 Max Chernoff

\wlog{lua-widow-control v3.0.0} %%version

\ifx\directlua\undefined
    \errmessage{%
        LuaTeX is required for this package.
        Make sure to compile with `luatex'%
    }
\fi

\catcode`@=11

% We need to change some code if we're using LuaMetaTeX
\def\lwc@iflmtx{\ifnum\luatexversion>200\relax}

\input ltluatex % \LuaTeX{}Base

\clubpenalty=1
\widowpenalty=1
\displaywidowpenalty=1
\brokenpenalty=1

\newdimen\lwcemergencystretch
\lwcemergencystretch=3em

\newdimen\lwcdraftoffset
\lwcdraftoffset=1in

\newcount\lwcmaxcost
\lwcmaxcost=2147483647

\directlua{require "lua-widow-control"}

% Here, we enable font expansion/contraction. It isn't strictly necessary for
% \lwc/'s functionality; however, it is required for the
% lengthened paragraphs to not have terrible spacing.
\lwc@iflmtx\else
    \expandglyphsinfont\the\font 20 20 5
    \adjustspacing=2
\fi
```

```
% Enable \lwc/ by default when the package is loaded.
\lwc@enable

% Fix some strange LMTX bugs
\lwc@iflmtx
    \normalizelinemode=\numexpression\normalizelinemode bor 2\relax
\fi

% Expansion of some parts of the document, such as section headings, is quite
% undesirable, so we'll disable \lwc/ for certain commands.

% We should only reenable \lwc/ at the end if it was already enabled.
\newcount\lwc@disable@count

\def\lwc@patch@pre{%
    \lwc@if@enabled%
        \advance\lwc@disable@count by 1%
        \lwc@disable%
    \fi%
}

\def\lwc@patch@post{
    \ifnum\lwc@disable@count>0%
        \lwc@enable%
        \advance\lwc@disable@count by -1%
    \fi
}

\def\lwc@extractcomponents #1:#2->#3\STOP{%
    \def\lwc@params{#2}%
    \def\lwc@body{#3}%
}

\def\lwcdisablecmd#1{%
    \ifdefined#1%
        \expandafter\lwc@extractcomponents\meaning#1\STOP%
        \begingroup%
            \catcode`@=11%
            \expanded{%
                \noexpand\scantokens{%
                    \gdef\noexpand#1\lwc@params{%
                        \noexpand\lwc@patch@pre\lwc@body\noexpand\lwc@patch@post%
                    }%
                }%
            }%
```

```
        \endgroup%
    \fi%
}

\begingroup
    \lwc@iflmtx\else
        \suppressoutererror=1
    \fi
    \lwcdisablecmd{\beginsection} % Sectioning
\endgroup

% Make the commands public
\let\lwcenable=\lwc@enable
\let\lwcdisable=\lwc@disable
\let\lwcdebug=\lwc@debug
\def\lwcdraft#1{%
    \lwc@show@costs{#1}%
    \lwc@show@colours{#1}%
}
\let\lwcshowcosts=\lwc@show@costs
\let\lwcshowcolours=\lwc@show@colours
\let\iflwc=\lwc@if@enabled
\let\lwcnobreak=\lwc@nobreak
\let\lwcpreshipout=\lwc@pre@shipout

\catcode`@=12
\endinput
```

lua-widow-control.sty

```
% lua-widow-control
% https://github.com/gucci-on-fleek/lua-widow-control
% SPDX-License-Identifier: MPL-2.0+
% SPDX-FileCopyrightText: 2022 Max Chernoff

% Formats built after 2015 include \LuaTeX{}Base, so this is the absolute
% minimum version that we will run under.
\NeedsTeXFormat{LaTeX2e}[2015/01/01]

% For _really_ old formats
\providecommand\DeclareRelease[3]{}
\providecommand\DeclareCurrentRelease[2]{}

\DeclareRelease{}{0000-00-00}{lua-widow-control-2022-02-22.sty}
\DeclareRelease{v1.1.6}{2022-02-22}{lua-widow-control-2022-02-22.sty}
\DeclareCurrentRelease{v3.0.0}{2022-11-22} %%version %%dashdate

% If this version of LaTeX doesn't support command hooks, then we load
% the last v1.1.X version of the package.
\providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
\IfFormatAtLeastTF{2020/10/01}{}{\input{lua-widow-control-2022-02-22.sty}}
\IfFormatAtLeastTF{2020/10/01}{}{\endinput}

\ProvidesExplPackage
    {lua-widow-control}
    {2022/11/22} %%slashdate
    {v3.0.0} %%version
    {Use Lua to remove widows and orphans}

% Message and String Constants
\str_const:Nn \c__lwc_name_str { lua-widow-control }

\msg_new:nnn
    { \c__lwc_name_str }
    { no-luatex }
    {
        LuaTeX~ is~ REQUIRED! \\
        Make~ sure~ to~ compile~ your~ document~ with~ `lualatex'.
    }

\msg_new:nnn
    { \c__lwc_name_str }
    { patch-failed }
    {
```

```
        Patching~ \c_backslash_str #1~ failed. \\
        Please~ ensure~ that~ \c_backslash_str #1~ exists.
    }

\msg_new:nnn
    { \c__lwc_name_str }
    { old-format-patch }
    {
        Patching~ not~ supported~ with~ old~ LaTeX. \\
        Please~ use~ a~ LaTeX~ format~ >=~ 2021/06/01.
    }

\msg_new:nnn
    { \c__lwc_name_str }
    { old-command }
    {
        \c_backslash_str #1~ has~ been~ REMOVED! \\
        Please~ use~ \c_backslash_str setuplwc \c_left_brace_str #2
        \c_right_brace_str\ instead.
    }

% Don't let the user proceed unless they are using \LuaTeX{}.
\sys_if_engine_luatex:F {
    \msg_critical:nn { \c__lwc_name_str } { no-luatex }
}

% Define (most of) the keys
\cs_generate_variant:Nn \keys_define:nn { Vn }

\keys_define:Vn { \c__lwc_name_str } {
    emergencystretch .dim_gset:N       = \g__lwc_emergencystretch_dim,
    emergencystretch .value_required:n = true,
    emergencystretch .initial:x        = \dim_max:nn { 3em } { 30pt },

    draftoffset .dim_gset:N       = \g__lwc_draftoffset_dim,
    draftoffset .value_required:n = true,
    draftoffset .initial:x        = 1in,

    max-cost .int_gset:N       = \g__lwc_maxcost_int,
    max-cost .value_required:n = true,
    max-cost .initial:x        = \c_max_int,

    widowpenalty .code:n = \int_gset:Nn \tex_widowpenalty:D        { #1 }
                           \int_gset:Nn \tex_displaywidowpenalty:D { #1 },
    widowpenalty .value_required:n = true,
```

```
    widowpenalty .initial:n        = 1,

    orphanpenalty .code:n = \int_gset:Nn \tex_clubpenalty:D  { #1 }
                            \int_gset:Nn \@clubpenalty       { #1 },
    orphanpenalty .value_required:n = true,
    orphanpenalty .initial:n        = 1,

    brokenpenalty .int_gset:N       = \tex_brokenpenalty:D,
    brokenpenalty .value_required:n = true,
    brokenpenalty .initial:n        = 1,

    microtype .bool_gset:N      = \g__lwc_use_microtype_bool,
    microtype .value_required:n = true,
    microtype .initial:n        = true,
    microtype .usage:n          = preamble,

    disablecmds .clist_gset:N     = \g__lwc_disablecmds_cl,
    disablecmds .value_required:n = false,
    disablecmds .initial:n        = { \@sect,           % LaTeX default
                                      \@ssect,          % LaTeX starred
                                      \M@sect,          % Memoir
                                      \@mem@old@ssect,  % Memoir Starred
                                      \ttl@straight@ii, % titlesec normal
                                      \ttl@top@ii,      % titlesec top
                                      \ttl@part@ii,     % titlesec part
                                    },
    disablecmds .usage:n          = preamble,
}

% Load the Lua code
\cs_if_exist:NTF \lua_load_module:n {
    \lua_load_module:n { lua-widow-control }
} {
    \lua_now:n { require "lua-widow-control" }
}

% Here, we enable font expansion/contraction. It isn't strictly necessary for
% \lwc/'s functionality; however, it is required for the
% lengthened paragraphs to not have terrible spacing.
\hook_gput_code:nnn { begindocument / before } { \c__lwc_name_str } {
    \bool_if:NT \g__lwc_use_microtype_bool {
        \@ifpackageloaded { microtype } {} {
            \RequirePackage[
                final,
                activate = { true, nocompatibility }
```

```
            ]
            { microtype }
        }
    }
}

% Core Function Definitions
\cs_new_eq:NN \iflwc \__lwc_iflwc:

\prg_new_conditional:Nnn \__lwc_if_enabled: { T, F, TF } {
    \__lwc_if_enabled:
        \prg_return_true:
    \else
        \prg_return_false:
    \fi
}

\prg_new_conditional:Nnn \__lwc_if_lmtx: { T, F, TF } {
    \int_compare:nNnTF { \tex_luatexversion:D } > { 200 } {
        \prg_return_true:
    } {
        \prg_return_false:
    }
}

% Expansion of some parts of the document, such as section headings, is quite
% undesirable, so we'll disable \lwc/ for certain commands.
\int_new:N \g__lwc_disable_int

\cs_new:Npn \__lwc_patch_pre: {
    % We should only reenable \lwc/ at the end if it was already enabled.
    \__lwc_if_enabled:T {
        \int_gincr:N \g__lwc_disable_int
        \__lwc_disable:
    }
}

\cs_new:Npn \__lwc_patch_post: {
    \int_compare:nT { \g__lwc_disable_int > 0 } {
        \__lwc_enable:
        \int_gdecr:N \g__lwc_disable_int
    }
}

\cs_new:Npn \__lwc_patch_cmd:c #1 {
```

```
    \IfFormatAtLeastTF { 2021/06/01 } {
        \hook_gput_code:nnn { cmd / #1 / before } { \c__lwc_name_str } {
            \__lwc_patch_pre:
        }
        \hook_gput_code:nnn { cmd / #1 / after } { \c__lwc_name_str } {
            \__lwc_patch_post:
        }
    } {
        \msg_warning:nn
                { \c__lwc_name_str }
                { old-format-patch }
    }
}

\cs_new:Npn \__lwc_patch_cmd:N #1 {
    \__lwc_patch_cmd:c { \cs_to_str:N #1 }
}

\cs_new:Npn \__lwc_patch_cmd:n #1 {
    % If the item provided is a single token, we'll assume that it's a \macro.
    % If it is multiple tokens, we'll assume that it's a `csname`.
    \tl_if_single:nTF { #1 } {
        \__lwc_patch_cmd:c { \cs_to_str:N #1 }
    } {
        \__lwc_patch_cmd:c { #1 }
    }
}

\hook_gput_code:nnn { begindocument / before } { \c__lwc_name_str } {
    \clist_map_function:NN \g__lwc_disablecmds_cl \__lwc_patch_cmd:n
}

\__lwc_if_lmtx:T {
    \int_gset:Nn \normalizelinemode {
        \numexpression\normalizelinemode bor 2\relax
    }
}

%%% Class and package-specifc patches

% KOMA-Script
\cs_if_exist:NT \AddtoDoHook {
    \AddtoDoHook { heading / begingroup } { \__lwc_patch_pre:  \use_none:n }
    \AddtoDoHook { heading / endgroup   } { \__lwc_patch_post: \use_none:n }
}
```

```
% Memoir
\cs_gset_nopar:Npn \pen@ltyabovepfbreak { 23 } % Issue #32

% Define some final keys
\keys_define:Vn { \c__lwc_name_str } {
    enable .choice:,
    enable / true  .code:n     = \__lwc_enable:,
    enable / false .code:n     = \__lwc_disable:,
    enable .initial:n          = true,
    enable .default:n          = true,
    enable .value_required:n   = false,

    disable .code:n            = \__lwc_disable:,
    disable .value_forbidden:n = true,

    debug .choice:,
    debug / true  .code:n     = \__lwc_debug:n { true  },
    debug / false .code:n     = \__lwc_debug:n { false },
    debug .default:n          = true,
    debug .value_required:n   = false,

    showcolours .choice:,
    showcolours / true  .code:n     = \__lwc_show_colours:n { true  },
    showcolours / false .code:n     = \__lwc_show_colours:n { false },
    showcolours .default:n          = true,
    showcolours .value_required:n   = false,

    showcosts .choice:,
    showcosts / true  .code:n     = \__lwc_show_costs:n   { true  },
    showcosts / false .code:n     = \__lwc_show_costs:n   { false },
    showcosts .default:n          = true,
    showcosts .value_required:n   = false,

    draft .meta:n   = {
        showcolours = { #1 },
        showcosts   = { #1 },
    },
    draft .default:n          = true,
    draft .value_required:n   = false,

    nobreak .code:n           = \__lwc_nobreak:n { #1 },
    nobreak .value_required:n = true,
    nobreak .initial:n        = keep,

    strict .meta:n = { emergencystretch = 0pt,
```

```
                        max-cost         = 5000,
                        nobreak          = warn,
                        widowpenalty     = 1,
                        orphanpenalty    = 1,
                        brokenpenalty    = 1,
                      },
    strict .value_forbidden:n = true,

    default .meta:n = { emergencystretch = 3em,
                        max-cost         = \c_max_int,
                        nobreak          = keep,
                        widowpenalty     = 1,
                        orphanpenalty    = 1,
                        brokenpenalty    = 1,
                      },
    default .value_forbidden:n = true,

    balanced .meta:n = { emergencystretch = 1em,
                         max-cost         = 10000,
                         nobreak          = keep,
                         widowpenalty     = 500,
                         orphanpenalty    = 500,
                         brokenpenalty    = 500,
                       },
    balanced .value_forbidden:n = true,
}

% Add the user interface for the keys
\IfFormatAtLeastTF { 2022-06-01 } {
    \ProcessKeyOptions [ \c__lwc_name_str ]
}{
    \RequirePackage { l3keys2e }
    \exp_args:NV \ProcessKeysOptions { \c__lwc_name_str }
}

\cs_generate_variant:Nn \keys_set:nn { Vn }
\NewDocumentCommand \lwcsetup {m} {
    \keys_set:Vn { \c__lwc_name_str }{ #1 }
}

% Legacy Commands
\NewDocumentCommand \lwcemergencystretch { } {
    \msg_error:nnnnn
        { \c__lwc_name_str }
```

```
        { old-command }
        { lwcemergencystretch }
        { emergencystretch=XXXpt }
}

\NewDocumentCommand \lwcdisablecmd { m } {
    \msg_error:nnxx
        { \c__lwc_name_str }
        { old-command }
        { lwcdisablecmd }
        { disablecmds={\c_backslash_str aaa,~ \c_backslash_str bbb} }
}

\cs_new_eq:NN \lwcenable  \__lwc_enable:
\cs_new_eq:NN \lwcdisable \__lwc_disable:

\endinput
```

lua-widow-control-2022-02-22.sty

```
% lua-widow-control
% https://github.com/gucci-on-fleek/lua-widow-control
% SPDX-License-Identifier: MPL-2.0+
% SPDX-FileCopyrightText: 2022 Max Chernoff

\NeedsTeXFormat{LaTeX2e}[2015/01/01] % Formats built after 2015 include \LuaTeX{}Base
\ProvidesPackage{lua-widow-control}%
    [2022/02/22 v1.1.6]

% The version number above is somewhat-misleading: I will make bugfixes to this file
% from time to time, but the core of the file will not change. Therefore, we should
% report a real version number here for debugging.
\PackageInfo{lua-widow-control}{%
    Real version:
    2022/11/22 %%slashdate
    v3.0.0 %%version
}

\PackageWarning{lua-widow-control}{%
    Old LaTeX format detected!\MessageBreak\MessageBreak
    Lua-widow-control prefers a LaTeX format\MessageBreak
    newer than November 2020. I'll still run\MessageBreak
    the latest Lua code, but I'm using an older\MessageBreak
    version of the LaTeX code. This means that\MessageBreak
    the key-value interface is *UNSUPPORTED*.\MessageBreak
}

\ifdefined\directlua\else
    \PackageError{lua-widow-control}{%
        LuaTeX is required for this package.\MessageBreak
        Make sure to compile with `lualatex'%
    }{}
\fi

\clubpenalty=1
\widowpenalty=1
\displaywidowpenalty=1

% We can't use \\newlength since that makes a \TeX{} "skip", not a "dimen"
\newdimen\lwcemergencystretch
\lwcemergencystretch=3em

\newcount\lwcmaxcost
\lwcmaxcost=2147483647
```

```
\directlua{require "lua-widow-control"}

% Here, we enable font expansion/contraction. It isn't strictly necessary for
% \lwc/'s functionality; however, it is required for the
% lengthened paragraphs to not have terrible spacing.
\RequirePackage{etoolbox}
\AtEndPreamble{
    \@ifpackageloaded{microtype}{}{ % Only load if not already loaded
        \RequirePackage[
            final,
            activate={true,nocompatibility}
        ]{microtype}
    }
}

% Define \TeX{} wrappers for Lua functions
\newcommand{\lwcenable}{\directlua{lwc.enable_callbacks()}}
\newcommand{\lwcdisable}{\directlua{lwc.disable_callbacks()}}
\newcommand{\iflwc}{\directlua{lwc.if_lwc_enabled()}}

% Enable \lwc/ by default when the package is loaded.
\lwcenable

% Expansion of some parts of the document, such as section headings, is quite
% undesirable, so we'll disable \lwc/ for certain commands.
\newcommand{\lwc@patch@warning}[1]{\PackageWarning{lua-widow-control}{%
    Patching the \protect#1 command failed%
}}

% We should only reenable \lwc/ at the end if it was already enabled.
\newif\iflwc@should@reenable

\newcommand{\lwc@patch@pre}{%
    \iflwc%
        \lwc@should@reenabletrue%
        \lwcdisable%
    \else%
        \lwc@should@reenablefalse%
    \fi%
}

\newcommand{\lwc@patch@post}{%
    \iflwc@should@reenable%
        \lwcenable%
    \fi%
```

```
}

\newcommand{\lwcdisablecmd}[1]{%
    \ifdefined#1
        \pretocmd{#1}{\lwc@patch@pre}{}{\lwc@patch@warning{#1}}%
        \apptocmd{#1}{\lwc@patch@post}{}{\lwc@patch@warning{#1}}%
    \fi
}

\lwcdisablecmd{\@sect} % Sectioning

\endinput
```

t-lua-widow-control.mkxl

```
%D \module
%D   [     file=t-lua-widow-control,
%D      version=3.0.0, %%version
%D        title=lua-widow-control,
%D     subtitle=\ConTeXt module for lua-widow-control,
%D       author=Max Chernoff,
%D         date=2022-11-22, %%dashdate
%D    copyright=Max Chernoff,
%D      license=MPL-2.0+,
%D          url=https://github.com/gucci-on-fleek/lua-widow-control]
\startmodule[lua-widow-control]
\unprotect

% Preliminaries
\installnamespace{lwc}

\installswitchcommandhandler \????lwc {lwc} \????lwc

% Set up the options
\newdimen\lwc_emergency_stretch
\newdimen\lwc_draft_offset
\newcount\lwc_max_cost

\starttexdefinition lwc_set_parameters
    \lwc_emergency_stretch=\lwcparameter{emergencystretch}
    \lwc_draft_offset=\lwcparameter{draftoffset}

    \doifelse{\lwcparameter{\c!state}}\v!start{
        \lwc_enable
    }{
        \lwc_disable
    }

    \lwc_debug{\lwcparameter{debug}}

    \doif{\lwcparameter{draft}}\v!start{
        \setlwcparameter{showcosts}{\v!start}
        \setlwcparameter{showcolours}{\v!start}
    }

    \doif{\lwcparameter{draft}}\v!stop{
        \setlwcparameter{showcosts}{\v!stop}
        \setlwcparameter{showcolours}{\v!stop}
    }
```

```
    \lwc_show_costs{\lwcparameter{showcosts}}
    \lwc_show_colours{\lwcparameter{showcolours}}

    \lwc_nobreak{\lwcparameter{nobreak}}

    \lwc_max_cost=\lwcparameter{maxcost}

    % We can't just set the penalties because they will be reset automatically
    % at \\starttext.
    \startsetups[*default]
        \directsetup{*reset}

        \clubpenalty=\lwcparameter{orphanpenalty}
        \widowpenalty=\lwcparameter{widowpenalty}
        \displaywidowpenalty=\lwcparameter{widowpenalty}
        \brokenpenalty=\lwcparameter{brokenpenalty}
    \stopsetups

    \startsetups[grid][*default]
        \directsetup{*reset}

        \clubpenalty=\lwcparameter{orphanpenalty}
        \widowpenalty=\lwcparameter{widowpenalty}
        \displaywidowpenalty=\lwcparameter{widowpenalty}
        \brokenpenalty=\lwcparameter{brokenpenalty}
    \stopsetups

    \setups[*default]
\stoptexdefinition

% Load the main Lua file
\ctxloadluafile{lua-widow-control}

% Define the presets
\definelwc[default][
    emergencystretch=3em,
    maxcost=2147483647,
    nobreak=keep,
    orphanpenalty=1,
    widowpenalty=1,
    brokenpenalty=1,
]

\definelwc[strict][
    emergencystretch=0pt,
    maxcost=5000,
```

```
    nobreak=warn,
    widowpenalty=1,
    orphanpenalty=1,
    brokenpenalty=1,
]

\definelwc[balanced][
    emergencystretch=1em,
    maxcost=10000
    nobreak=keep,
    widowpenalty=500,
    orphanpenalty=500,
    brokenpenalty=500,
]

% Set up the default options
\setuplwc[
    \c!state=\v!start,
    debug=\v!stop,
    draft=,
    showcosts=\v!stop,
    showcolours=\v!stop,
    draftoffset=1in,
]

\setuplwc[default]

\appendtoks
    \ifcase\lwcsetupmode
        % can't happen
    \or % \setuplwc[name][key=value]
      \ifx\previouslwc\currentlwc
        \let\currentlwc\currentlwc
        \lwc_set_parameters
      \fi
    \or % \setuplwc[key=value]
      \let\currentlwc\previouslwc
      \lwc_set_parameters
    \or % \setuplwc[name]
      \glet\currentlwc\currentlwc % global
      \lwc_set_parameters
    \fi
\to \everysetuplwc

\lwc_set_parameters
```

```
% Here, we enable font expansion/contraction. It isn't strictly necessary for
% \lwc/'s functionality; however, it is required for the
% lengthened paragraphs to not have terrible spacing.
\definefontfeature[default][default][expansion=quality]
\setupalign[hz]

% Expansion of some parts of the document, such as section headings, is quite
% undesirable, so we'll disable \lwc/ for certain commands.
% We should only reenable \lwc/ at the end if it was already enabled.
\newcount\lwc_disable_count

\define\lwc_patch_pre{%
    \lwc_if_enabled%
        \advance\lwc_disable_count by 1%
        \setuplwc[\c!state=\v!stop]%
    \fi%
}

\define\lwc_patch_post{
    \ifnum\lwc_disable_count>0\relax%
        \setuplwc[\c!state=\v!start]%
        \advance\lwc_disable_count by -1%
    \fi%
}

% Add the default patches
\prependtoks\lwc_patch_pre\to\everybeforesectionheadhandle % Sectioning
\prependtoks\lwc_patch_post\to\everyaftersectionheadhandle

% Make the commands public
\let\iflwc=\lwc_if_enabled

\protect
\stopmodule
```

lua-widow-control.opm

```
% lua-widow-control
% https://github.com/gucci-on-fleek/lua-widow-control
% SPDX-License-Identifier: MPL-2.0+
% SPDX-FileCopyrightText: 2022 Max Chernoff

\_codedecl\lwcenable{lua-widow-control <v3.0.0>} %%version
\_namespace{lwc}

\_clubpenalty=1
\_widowpenalty=1
\_displaywidowpenalty=1
\_brokenpenalty=1

\_newdimen\lwcemergencystretch
\lwcemergencystretch=3em

\_newdimen\lwcdraftoffset
\lwcdraftoffset=1in

\_newcount\lwcmaxcost
\lwcmaxcost=2147483647

\_directlua{require "lua-widow-control"}

% Enable \lwc/ by default when the package is loaded.
\.enable

% Expansion of some parts of the document, such as section headings, is quite
% undesirable, so we'll disable \lwc/ for certain commands.

% We should only reenable \lwc/ at the end if it was already enabled.
\_newcount\.disable_count

\_def\.patch_pre{%
    \.if_enabled%
        \_advance\.disable_count by 1%
        \.disable%
    \_fi%
}

\_def\.patch_post{
    \_ifnum\.disable_count>0%
        \.enable%
        \_advance\.disable_count by -1%
    \_fi
```

```
}

\_def\.extractcomponents #1:#2->#3\STOP{%
    \_def\.params{#2}%
    \_def\.body{#3}%
}

\def\.disable_cmd#1{%
    \_ifdefined#1%
        \_ea\.extractcomponents\_meaning#1\STOP%
        \_begingroup%
            \_catcode`_=11%
            \_expanded{%
                \_noexpand\_scantokens{%
                    \_gdef\_noexpand#1\.params{%
                        \_noexpand\.patch_pre\.body\_noexpand\.patch_post%
                    }%
                }%
            }%
        \_endgroup%
    \_fi%
}

\.disable_cmd{\_printchap}
\.disable_cmd{\_printsec}
\.disable_cmd{\_printsecc}

% Make the commands public
\_let\lwcenable=\.enable
\_let\lwcdisable=\.disable
\_let\lwcdisablecmd=\.disable_cmd
\_let\lwcdebug=\.debug
\_def\lwcdraft#1{%
    \.show_costs{#1}%
    \.show_colours{#1}%
}
\_let\lwcshowcosts=\.show_costs
\_let\lwcshowcolours=\.show_colours
\_let\iflwc=\.if_enabled
\_let\lwcnobreak=\.nobreak

\_endnamespace
\_endcode
```

Demo from Table 1

```
\definepapersize[smallpaper][
    width=6cm,
    height=8.3cm
]\setuppapersize[smallpaper]

\setuplayout[
    topspace=0.1cm,
    backspace=0.1cm,
    width=middle,
    height=middle,
    header=0pt,
    footer=0pt,
]

\def\lwc/{\sans{lua-\allowbreak widow-\allowbreak control}}
\def\Lwc/{\sans{Lua-\allowbreak widow-\allowbreak control}}

\setupbodyfont[9pt]
\setupindenting[yes, 2em]

\definepalet[layout][grid=middlegray]
\showgrid[nonumber, none, lines]

\definefontfeature[default][default][expansion=quality,protrusion=quality]

\usetypescript[modern-base]
\setupbodyfont[reset,modern]

\setupalign[hz,hanging,tolerant]

\setuplanguage[en][spacing=packed]

\starttext
    \Lwc/ can remove most widows and orphans from a document, \emph{without} stretching
      any glue or shortening any pages.

    It does so by automatically lengthening a paragraph on a page where a widow or
      orphan would otherwise occur. While \TeX{} breaks paragraphs into their natural
      length, \lwc/ is breaking the paragraph 1~line longer than its natural length.
      \TeX{}'s paragraph is output to the page, but \lwc/'s paragraph is just stored
      for later. When a widow or orphan occurs, \lwc/ can take over. It selects the
      previously-saved paragraph with the least badness; then, it replaces \TeX{}'s
      paragraph with its saved paragraph. This increases the text block height of the
      page by 1~line.
```

```
    Now, the last line of the current page can be pushed to the top of the next page.
      This removes the widow or the orphan without creating any additional work.
\stoptext
```