LUA-WIDOW-CONTROL

Max Chernoff

v 2.1.0

---

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 your document. For more details, see the **Usage sections**.

CONTENTS

MOTIVATION

TEX provides top-notch typesetting: even 40 years after its first release, no other program produces higher quality mathematical typesetting, and its paragraph-breaking algorithm is still state-of-the-art. However, its page breaking is not quite as sophisticated as its paragraph breaking and thus suffers from some minor issues.

Unmodified TEX typically has only 2 ways of dealing with widows and orphans: it can either shorten a page by 1 line, or it can stretch out some 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 behavior works quite well; however, for prose or

any other document composed almost entirely of text, there is no vertical whitespace to stretch.

Since there were no ready-made and fully-automated solutions to remove widows and orphans from all types of documents, I decided to create lua-widow-control.

WIDOWS AND ORPHANS

Widows Widows occur when when the majority of a paragraph is on one page or column, but the last line is on the following page or column. Widows are undesirable for both aesthetics and readability. Aesthetically, it looks quite odd for a lone line to be at the start of the page. Functionally, the separation of a paragraph and its last line disconnects the two, causing the reader to lose context for the widowed line.

Orphans Orphans are when the first line of a paragraph occurs on the page or column before the remainder of they paragraph. They are not nearly 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 do, so they tend to be ignored more often.

Broken Hyphens "Broken" hyphens occur whenever a page break occurs part way through a hyphenated word. These really have nothing to do with widows and orphans; however, TEX treats broken hyphens exactly the same as it does widows and orphans. In addition, breaking a word across two pages is at almost as bad for the reader as widows and orphans; therefore, lua-widow-control treats broken hyphens exactly the same way as it treats widows and orphans.

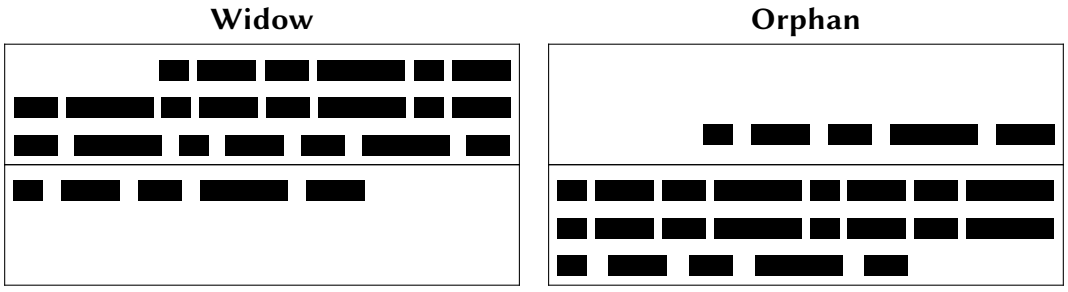

Figure 1 A visual comparison of widows and orphans.

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. Chapter 15 of *The TEXBook*[1] ("How TEX Makes Lines into Pages") is the best reference for this; however, it goes into much more detail than most users require. As a supplemental resource, I can also recommend Section 27 of *TEXby Topic*[2], **available online** for free. Below follows a *very* simplified (and likely error-ridden) 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 vertical spaces.

However, some objects have penalties attached. These penalties make TEX treat the object as if it is longer or shorter for the sake of page breaking. By default, TEX assigns a penalties to the first and last lines of a paragraph (widows and orphans). This makes TEX treat them as if they are larger or smaller than their actual size such that TEX tends not to break them up.

One important note: once TEX begins breaking a page, it never goes back and modifies any content on the page. Page breaking is a localized algorithm, without any backtracking.

Behavior

Of course, this algorithm doesn't allow us to intuitively understand how TEX deals with widows and orphans.

Due to the penalties attached to widows and orphans, TEX treats them as if they are longer than they actually are. Widows and orphans with small penalties attached—like LATEX's default values of 150—are only treated as slightly taller than 1 line, while widows and orphans with large penalties—values near 10 000—are treated as if they are 2 lines tall. Because potential widow and orphan lines are broken as if they are taller than they actually are, TEX will tend to group them together on the same pages.

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. The main takeaway is that for a page exclusively filled with text, all of TEX's builtin solutions come with compromises.

OTHER SOLUTIONS

There have been a few previous attempts to improve upon TEX's previously-discussed widow and orphan-handling abilities; however, none of these have been able to automatically remove widows and orphans without stretching any glue or shortening any pages.

Strategies against widows[3] and *Managing forlorn paragraph lines in LATEX*[4] both begin with comprehensive discussions of the methods of preventing widows and orphans. They both agree that widows and orphans are bad and ought to be avoided; however, they each differ in solutions. *Strategies*[3] proposes an output routine that reduces the length of facing pages by 1 line when necessary to remove widows and orphans while *Managing*[4] proposes that the author manually rewrites or adjusts the `\looseness` when needed.

Paragraph callback to help with widows/orphans hand tuning[5] contains a file `widow-assist.lua` that automatically detects which paragraphs can be safely shortened or lengthened by 1 line. *The widows-and-orphans package*[6] alerts the author to the pages that contain widows or orphans. Combined, these packages make it very simple for the author to quickly remove widows and orphans by adjusting the `\looseness` values; however, it still requires the author to make manual source changes after each revision.

lua-widow-control is essentially just a combination of `widow-assist.lua`[5] and widows-and-orphans:[6] when the `\outputpenalty` shows that a widow or orphan occurred, Lua is used to find a stretchable paragraph. What lua-widow-control adds on top of these packages is automation: lua-widow-control eliminates the requirement for any manual adjustments.

DEMONSTRATION

Although TEX's page breaking algorithm is quite simple, it can lead to some fairly complex behaviors when widows and orphans are involved. The usual choices are to either ignore them, stretch some glue, or shorten the page. **Table 1** has a visual demonstration of some of these behaviors and how lua-widow-control differs from the defaults.

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 pretty distracting for the reader, so it is best avoided wherever possible.

Ignore	Shorten	Stretch	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 lengthening 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-	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 lengthening 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.	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 lengthening 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.	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 lengthening 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.
out creating any additional work.	This removes the widow or the orphan without creating any additional work.	This removes the widow or the orphan without creating any additional work.	This removes the widow or the orphan without creating any additional work.

`\parskip=0pt`
 	
`\clubpenalty=0`
`\widowpenalty=0`
	`\parskip=0pt`
	
	`\clubpenalty=10000`
	`\widowpenalty=10000`
		`\parskip=0pt plus 1fill`
		
		`\clubpenalty=10000`
		`\widowpenalty=10000`
			`\usepackage{lua-widow-control}`

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

Shorten This page did not leave any widows, but it did shorten the previous page by 1 line. Sometimes this is acceptable, but usually it looks bad because each page will 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 flushed bottom margin. However, the space between each paragraph 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 behavior. However, in a page with mostly text, these paragraph gaps can look unsightly.

In addition, this method is incompatible with typesetting on a grid since all glue stretch must be quantized to the height of a line.

lua-widow-control Lua-widow-control has none of these issues: it eliminates the widows in a document while keeping a flushed 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 it becomes nearly 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.

ConTEXt MKIV Standalone Run `first-setup.sh --modules="lua-widow-control"` in a terminal or install manually.

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** or **GitHub**.
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 certainly 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. Instead, you should follow 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.

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 always use the `\setuplwc[`⟨*options*⟩`]` command.

Plain TEX and OpTEX are a little different. Some options have commands provided (i.e., `\lwcemergencystretch = `⟨*dimension*⟩), while others must be set manually (i.e., `\directlua{lwc.debug = true}`).

Also, please note that not all commands are provided for all formats.

Enabling

Lua-widow-control is enabled by default as soon as you load it. Nevertheless, you may need to explicitly reenable it if you have previously disabled it.

Plain TEX/OpTEX	`\lwcenable`
LATEX	`\lwcsetup{enable}`
ConTEXt	`\setuplwc[state = start]`

Disabling

Plain TEX/OpTEX	`\lwcdisable`
LATEX	`\lwcsetup{disable}`
ConTEXt	`\setuplwc[state = stop]`

Strict Mode

By default, lua-widow-control takes all possible measures to remove widows and orphans. This normally works out pretty well; however, sometimes these measures may be a little more aggressive than certain users want.

Lua-widow-control offers a "strict" mode that will only make modification to the page that are near-imperceptible to remove widows and orphans.

LaTeX `\lwcsetup{strict}`

Internally, this sets `emergencystretch = 0pt`, `max-cost = 5000`, and `no-break = warn`.

`\emergency stretch`

You can configure the `\emergencystretch` used when stretching a paragraph. The default value is 3 em.

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 still 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⟩]`

Selectively Disabling

Sometimes, you may want to disable lua-widow-control for certain commands where stretching is undesirable. For example, you typically wouldn't want section headings to be stretched.

You could just disable then reenable lua-widow-control every time that you use the command; however, lua-widow-control provides a convenience macro that will do this automatically for you.

Lua-widow-control automatically patches the default LaTeX, ConTeXt, Plain TeX, and OpTeX section commands, so you shouldn't need to patch these yourself. Lua-widow-control also patches the commands provided by memoir, KOMA-script, and titlesec. You'll need to patch any other section commands yourself.

Note that the LaTeX option does not *append* commands to the disable list; rather, it *sets* the list from scratch. This means that you can set an empty list to disable any command patching; however, this also means that if you want to disable one command on top of the defaults, you'll also need to manually include the default list.

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⟩`

Widow and Orphan Penalties

You can also manually adjust the penalties that TeX assigns to widows, orphans, and broken hyphens. Usually, the defaults are fine, but advanced users 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⟩]`

Some suitable integers:

Ignore widows/orphans	0
Default	1
Disable lua-widow-control	10 000

`\nobreak` Behaviour

When lua-widow-control encounters an orphan, it removes it by removing the orphaned line to the next page. However, sometimes, an orphan is immediately preceded by a section heading or a `\nobreak` command. By moving the orphan to the next page, you would naïvely separate a section (or other such material) from the line that follows. This really ought to be avoided, so lua-widow-control provides some options to avoid 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 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.

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.

Maximum Cost

When TEX breaks a paragraph, it scores it by the number of "demerits". The demerits for a paragraph is the sum of the squared badnesses for each line, plus any "additional demerits" added for any other reason. The badness for a line is proportional the cube of the glue stretch ratio, so demerits grow with the sixth power of glue stretch.

To choose the "best" paragraph on the page, 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.

By default, lua-widow-control just 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⟩]`

Very advanced users may also set a custom cost function by redefining the `lwc.paragraph_cost(demerits, lines)` function.

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.

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

COLUMNS

Since TEX implements column breaking and page breaking through the same internal mechanisms, lua-widow-control should remove widows and orphans between columns just as well as it does with widows and orphans between pages. This has been tested with the standard LATEX class option `twocolumn` and the two-column output routine from Chapter 23 of *The TEXBook*.[1] Lua-widow-control should presumably work with any other multi-column implementation; however, due to the diversity and complexity of output routines, this cannot be guaranteed.

KNOWN ISSUES

- Lua-widow-control will rarely fail to correctly move the last line on an expanded page to the next page in documents with *very* small paper sizes. **(Issue #19)**
- When a 3-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 better than widows, so this is still an improvement.
- Sometimes a widow or orphan cannot be eliminated because no paragraph has enough "stretch". This can *sometimes* be remediated by increasing lua-widow-control's `\emergencystretch`; however, some pages just don't have enough "stretchy" paragraphs. Long paragraphs with short words tend to be "stretchier" than short paragraphs with long words since these long paragraphs will have

more interword glue. Narrow columns also stretch easier than wide columns since you need to expand a paragraph by less to make a new line.

- When running under luametaTEX, the log may be filled with lines like "`luatex warning > tex: left parfill skip is gone`". This is harmless and can be ignored. **(Issue #7)**
- Lua-widow-control will rarely raise a "`Circular node list detected!`" warning. This occurs when the replacement paragraph node list loops back on itself. This is usually harmless, but can rarely cause the entire compile to completely fail. **(Issue #9)**
- TEX may warn you about over or underfull vboxes on pages where lua-widow-control removed a widow or orphan. This is a false alarm and can be ignored. **(Issue #8)**
- If there is a footnote on the last line of the page with a widow or orphan, lua-widow-control will sometimes move the "footnote mark" but not the "footnote text", thus breaking up a footnote. **(Issue #26)**

THE ALGORITHM

Lua-widow-control uses a fairly simple algorithm to eliminate widows and orphans. It is pretty basic, but there are a few subtleties. Please see "**Implementation**" for a full listing of the source code.

Paragraph Breaking

First, lua-widow-control hooks into the paragraph breaking process.

Before a paragraph is broken by TEX, lua-widow-control grabs the unbroken paragraph. Lua-widow-control then breaks the paragraph 1 line longer than its natural length and stores it for later, *without* interfering with how TEX breaks paragraphs into their natural length.

After TEX has broken its paragraph into its natural length, lua-widow-control appears once again. Before the broken paragraph is added to the main vertical list, lua-widow-control tags the first and last nodes of the paragraph. These tags create a relationship between the previously-saved lengthened paragraph and the start/end of the naturally-typeset paragraph on the page.

Page Breaking

Lua-widow-control intercepts `\box255` immediately before the output routine.

First, lua-widow-control analyzes the `\outputpenalty` of the page or column. If the page was broken at a widow or orphan, the `\outputpenalty` will equal either `\widowpenalty` or `\orphanpenalty`. If the `\outputpenalty` is not indicative of a widow or orphan, lua-widow-control will stop and return `\box255` unmodified.

At this point, we know that we have a widow or orphan on the page, so we must lengthen the page by 1 line. We iterate through the list of saved paragraphs to find the lengthened paragraph with the least cost. Once we've selected a paragraph to replace, we can now traverse through the page to find the original version of this paragraph that TEX originally typeset. Once we find the original paragraph, we "splice" the lengthened paragraph in the place of the original.

Since the page is now 1 line longer than it was before, we pull the last line off of the page to bring it back to its original length. We place the line onto the top of the *recent contributions* list so that it is added to the start of the next page. Now, we can return the new, widow-free page to the output routine.

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. Knuth, DE (2020). *The TEXBook*. Addison–Wesley. `ctan.org/pkg/texbook`
2. Eijkhout, V (2007). *TEXby Topic*. Author. `texdoc.org/serve/texbytopic/0`
3. Isambert, P (2010). Strategies against widows. *TUGboat*, *31*(1), 12–17. `tug.org/TUGboat/tb31-1/tb97isambert.pdf`

4. Mittelbach, F (2018). Managing forlorn paragraph lines in LaTeX. *TUGboat*, *39*(3), 246–251. tug.org/TUGboat/tb39-3/tb123mitt-widows.pdf
5. jeremie (2017, August). *Paragraph callback to help with widows/orphans hand tuning*. tex.stackexchange.com/q/372062
6. Mittelbach, F (2021, March). *The widows-and-orphans package*. Author. ctan.org/pkg/widows-and-orphans

IMPLEMENTATION

lua-widow-control.lua

```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 prev node
--- @field next node
--- @field id integer
--- @field subtype integer
--- @field penalty integer
--- @field height integer
--- @field depth integer

-- Set some default variables
lwc = lwc or {}
lwc.name = "lua-widow-control"
lwc.nobreak_behaviour = "keep"

-- Locals for `debug_print`
local write_nl = texio.write_nl
local string_rep = string.rep
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_print(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{},
    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 format:find("cont") then -- cont-en, cont-fr, cont-nl, ...
    context = true
    if status.luatex_engine == "luametatex" then
        lmtx = true
    end
elseif format:find("latex") then -- lualatex, lualatex-dev, ...
    latex = true
elseif format == "luatex" then -- Plain
    plain = true
elseif format == "optex" then -- OpTeX
    optex = true
end

--[[
    Save some local copies of the node library to reduce table lookups.
    This is probably a useless micro-optimization, but it can't hurt.
  ]]
-- Node ID's
local baselineskip_subid = 2
local glue_id = node.id("glue")
local glyph_id = node.id("glyph")
local hlist_id = node.id("hlist")
local line_subid = 1
local linebreakpenalty_subid = 1
local par_id = node.id("par") or node.id("local_par")
local penalty_id = node.id("penalty")
```

```lua
-- Local versions of globals
local copy = node.copy_list or node.copylist
local find_attribute = node.find_attribute or node.findattribute
local flush_list = node.flush_list or node.flushlist
local free = node.free
local getattribute = node.get_attribute or node.getattribute
local insert_token = token.put_next or token.putnext
local last = node.slide
local new_node = node.new
local node_id = node.is_node or node.isnode
local set_attribute = node.set_attribute or node.setattribute
local string_char = string.char
local traverse = node.traverse
local traverseid = node.traverse_id or node.traverseid

-- Misc. Constants
local iffalse = token.create("iffalse")
local iftrue = token.create("iftrue")
local INFINITY = 10000
local min_col_width = tex.sp("250pt")
local SINGLE_LINE = 50
local PAGE_MULTIPLE = 100

--[[
    Package/module initialization
  ]]
local attribute,
      contrib_head,
      emergencystretch,
      info,
      max_cost,
      pagenum,
      stretch_order,
      warning

if lmtx then
    -- LMTX has removed underscores from most of the Lua parts
    debug_print("LMTX")
    contrib_head = "contributehead"
    stretch_order = "stretchorder"
else
    contrib_head = "contrib_head"
    stretch_order = "stretch_order"
end
```

```
if context then
    debug_print("ConTeXt")

    warning = logs.reporter(lwc.name, "warning")
    local _info = logs.reporter(lwc.name, "info")
    info = function (text)
        logs.pushtarget("logfile")
        _info(text)
        logs.poptarget()
    end
    attribute = attributes.public(lwc.name)
    pagenum = function() return tex.count["realpageno"] end

    -- Dimen names
    emergencystretch = "lwc_emergency_stretch"
    max_cost = "lwc_max_cost"
elseif plain or latex or optex then
    pagenum = function() return tex.count[0] end

    -- Dimen names
    if tex.isdimen("g__lwc_emergencystretch_dim") then
        emergencystretch = "g__lwc_emergencystretch_dim"
        max_cost = "g__lwc_maxcost_int"
    else
        emergencystretch = "lwcemergencystretch"
        max_cost = "lwcmaxcost"
    end

    if plain or latex then
        debug_print("Plain/LaTeX")
        luatexbase.provides_module {
            name = lwc.name,
            date = "2022/05/14", --%%dashdate
            version = "2.1.0", --%%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
        attribute = luatexbase.new_attribute(lwc.name)
```

```
    elseif optex then
        debug_print("OpTeX")

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

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

local paragraphs = {} -- List to hold the alternate paragraph versions

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

--[[
    Function definitions
  ]]

--- Prints the initial glyphs and glue of an hlist
--- @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 .. string_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_print(get_location(), chars)
end

--- The "cost function" to use. See the manual.
--- @param demerits number The demerits of the broken paragraph
--- @param lines number The number of lines in the broken paragraph
--- @return number The cost of the broken paragraph
function lwc.paragraph_cost(demerits, lines)
    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 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 traverseid(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
        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
end

--- Saves each paragraph, but lengthened by 1 line
---
--- Called by the `pre_linebreak_filter` callback
---
--- @param head node
--- @return node
function lwc.save_paragraphs(head)
    if (head.id ~= par_id and context) or -- Ensure that we were actually given a par
        status.output_active -- Don't run during the output routine
    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

    -- We need to return the unmodified head at the end, so we make a copy here
    local new_head = copy(head)

    -- Prevent ultra-short last lines (\TeX{}Book p. 104), except with narrow columns
    -- Equivalent to \\parfillskip=0pt plus 0.8\\hsize
    local parfillskip
    if lmtx or last(new_head).id ~= glue_id then
        -- LMTX does not automatically add the \\parfillskip glue
        parfillskip = new_node("glue", "parfillskip")
    else
        parfillskip = last(new_head)
    end

    if tex.hsize > min_col_width then
        parfillskip[stretch_order] = 0
        parfillskip.stretch = 0.8 * tex.hsize -- Last line must be at least 20% long
    end

    if lmtx or last(new_head).id ~= glue_id then
        last(new_head).next = parfillskip
```

```
end

-- Break the paragraph 1 line longer than natural
local long_node, long_info = tex.linebreak(new_head, {
    looseness = 1,
    emergencystretch = tex.getdimen(emergencystretch),
})

-- Break the natural paragraph so we know how long it was
nat_head = copy(head)

if lmtx then
    parfillskip = new_node("glue", "parfillskip")
    parfillskip[stretch_order] = 1
    parfillskip.stretch = 1 -- 0pt plus 1fil
    last(nat_head).next = parfillskip
end

local natural_node, natural_info = tex.linebreak(nat_head)
flush_list(natural_node)

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

if not grid_mode_enabled() then
    -- Offset the accumulated \\prevdepth
    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)

if long_info.prevgraf == natural_info.prevgraf + 1 and
   long_cost > 10 -- Any paragraph that is "free" to expand is suspicious
then
    table.insert(paragraphs, {
        cost = long_cost,
        node = next_of_type(long_node, hlist_id, { subtype = line_subid })
    })
end

-- Print some debugging information
get_chars(head)
debug_print(get_location(), "nat  lines    " .. natural_info.prevgraf)
```

```
    debug_print(
        get_location(),
        "nat  cost " ..
        lwc.paragraph_cost(natural_info.demerits, natural_info.prevgraf)
    )
    debug_print(get_location(), "long lines    " .. long_info.prevgraf)
    debug_print(
        get_location(),
        "long cost " ..
        lwc.paragraph_cost(long_info.demerits, long_info.prevgraf)
    )

    -- \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. Called by the
--- `post_linebreak_filter` callback.
---
--- @param head node
--- @return node
function lwc.mark_paragraphs(head)
    if not status.output_active then -- Don't run during the output routine
        -- Get the start and end of the paragraph
        local top_para = next_of_type(head, hlist_id, { subtype = line_subid })
        local bottom_para = last(head)

        if top_para ~= bottom_para then
            set_attribute(
                top_para,
                attribute,
                #paragraphs + (PAGE_MULTIPLE * pagenum())
            )
            set_attribute(
                bottom_para,
                attribute,
                -1 * (#paragraphs + (PAGE_MULTIPLE * pagenum()))
            )
        else
            -- We need a special tag for a 1-line paragraph since the node can only
```

```
            -- have one attribute value
            set_attribute(
                top_para,
                attribute,
                #paragraphs + (PAGE_MULTIPLE * pagenum()) + SINGLE_LINE
            )
        end
    end

    return head
end

--- A "safe" version of the last/slide function.
---
--- Sometimes the node list can form a loop. Since there is no last element
--- of a looped linked-list, the `last()` function will never terminate. This
--- function provides a "safe" version of the `last()` function that will break
--- the loop at the end if the list is circular. Called by the `pre_output_filter`
--- callback.
---
--- @param head node The start of a node list
--- @return node The last node in a list
local function safe_last(head)
    local ids = {}
    local prev

    while head.next do
        local id = node_id(head)

        if ids[id] then
            warning [[Circular node list detected!
This should never happen. I'll try and
recover, but your output may be corrupted.
(Internal Error)]]
            prev.next = nil
            debug_print("safe_last", node.type(head.id) .. " " .. node.type(prev.id))

            return prev
        end

        ids[id] = true
        head.prev = prev
        prev = head
        head = head.next
    end
```

```
    return head
end

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

    --[[
        We only need to process pages that have orphans or widows. If `paragraphs`
        is empty, then there is nothing that we can do.

        The list of penalties is from:
        https://tug.org/TUGboat/tb39-3/tb123mitt-widows-code.pdf#subsection.0.2.1
      ]]
    penalty = penalty - tex.interlinepenalty

    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

--- Remove the widows and orphans from the page, just after the output routine.
---
--- This function holds the "meat" of the module. It 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)
    local head_save = head -- Save the start of the `head` linked-list

    debug_print("outputpenalty", tex.outputpenalty .. " " .. #paragraphs)

    if not is_matching_penalty(tex.outputpenalty) or
       #paragraphs == 0
    then
        paragraphs = {}
        return head_save
    end

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

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

    local last_paragraph
    local head_last = last(head)
    -- Find the last paragraph on the page, starting at the end, heading in reverse
    while head_last do
        local value = getattribute(head_last, attribute)
        if value then
            last_paragraph = value % PAGE_MULTIPLE
            break
        end

        head_last = head_last.prev
    end

    local first_paragraph
    -- Find the first paragraph on the page, from the top
    local first_attribute_val, first_attribute_head = find_attribute(head, attribute)
    if first_attribute_val // 100 == pagenum() - 1 then
        -- If the first complete paragraph on the page was initially broken on the
        -- previous page, then we can't expand it here so we need to skip it.
        first_paragraph = find_attribute(
            first_attribute_head.next,
            attribute
```

```
    ) % PAGE_MULTIPLE
else
    first_paragraph = first_attribute_val % PAGE_MULTIPLE
end

-- We find the current "best" replacement, then free the unused ones
for i, paragraph in pairs(paragraphs) do
    if paragraph.cost < best_cost and
       i <  last_paragraph and
       i >= first_paragraph
    then
        -- Clear the old best paragraph
        flush_list(paragraphs[paragraph_index].node)
        paragraphs[paragraph_index].node = nil
        -- Set the new best paragraph
        paragraph_index, best_cost = i, paragraph.cost
    elseif i > 1 then
        -- Not sure why `i > 1` is required?
        flush_list(paragraph.node)
        paragraph.node = nil
    end
end

debug_print(
    "selected para",
    pagenum() ..
    "/" ..
    paragraph_index ..
    " (" ..
    best_cost ..
    ")"
)

if best_cost > tex.getcount(max_cost) or
   paragraph_index == last_paragraph
then
    -- If the best replacement is too bad, we can't do anything
    warning("Widow/Orphan/broken hyphen NOT removed on page " .. pagenum())
    paragraphs = {}
    return head_save
end

local target_node = paragraphs[paragraph_index].node

-- Start of final paragraph
```

```
debug_print("remove_widows", "moving last line")

-- Here we check to see if the widow/orphan was preceded by a large penalty
head = last(head_save).prev
local big_penalty_found, last_line, hlist_head
while head do
    if head.id == glue_id then
        -- Ignore any glue nodes
    elseif head.id == penalty_id and head.penalty >= INFINITY then
        -- Infinite break penalty
        big_penalty_found = true
    elseif big_penalty_found and head.id == hlist_id then
        -- Line before the penalty
        if lwc.nobreak_behaviour == "keep" then
            hlist_head = head
            big_penalty_found = false
        elseif lwc.nobreak_behaviour == "split" then
            head = last(head_save)
            break
        elseif lwc.nobreak_behaviour == "warn" then
            warning("Widow/Orphan/broken hyphen NOT removed on page " .. pagenum())
            paragraphs = {}
            return head_save
        end
    else
        -- Not found
        if hlist_head then
            head = hlist_head
        else
            head = last(head_save)
        end
        break
    end
    head = head.prev
end

local potential_penalty = head.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())
```

```
end

last_line = copy(head)
last(last_line).next = copy(tex.lists[contrib_head])

head.prev.prev.next = nil
-- Move the last line to the next page
tex.lists[contrib_head] = last_line

local free_next_nodes = false

-- Loop through all of the nodes on the page with the lwc attribute
head = head_save
while head do
    local value
    value, head = find_attribute(head, attribute)

    if not head then
        break
    end

    debug_print("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_print("remove_widows", "replacement start")
        safe_last(target_node) -- Remove any loops

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

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

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

```
            head.prev.next = target_node
            free_next_nodes = true
        end

        -- Insert the end of the replacement paragraph
        if value == -1 * (paragraph_index + (PAGE_MULTIPLE * pagenum())) or
           value ==       paragraph_index + (PAGE_MULTIPLE * pagenum()) + SINGLE_LINE
        then
            debug_print("remove_widows", "replacement end")
            local target_node_last = safe_last(target_node)

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

                after_glue.next = head.next
            else
                target_node_last.next = head.next
            end

            break
        end

        if free_next_nodes then
            head = free(head)
        else
            head = head.next
        end
    end

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

    paragraphs = {} -- Clear paragraphs array at the end of the page

    return head_save
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 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",
    }),
}
```

```
local enabled = false
--- Enable the paragraph callbacks
function lwc.enable_callbacks()
    debug_print("callbacks", "enabling")
    if not enabled then
        lwc.callbacks.save_paragraphs.enable()
        lwc.callbacks.mark_paragraphs.enable()

        enabled = true
    else
        info("Already enabled")
    end
end

--- Disable the paragraph callbacks
function lwc.disable_callbacks()
    debug_print("callbacks", "disabling")
    if enabled then
        lwc.callbacks.save_paragraphs.disable()
        lwc.callbacks.mark_paragraphs.disable()
        --[[
            We do \emph{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.
          ]]

        enabled = false
    else
        info("Already disabled")
    end
end

function lwc.if_lwc_enabled()
    debug_print("iflwc")
    if 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 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

        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

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" }
)

--- 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.
```

```
local function silence_luatexbase()
    local nups = debug.getinfo(luatexbase.add_to_callback).nups

    for i = 1, nups do
        local name, func = debug.getupvalue(luatexbase.add_to_callback, i)
        if name == "luatexbase_log" then
            debug.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

-- Activate \lwc/
if plain or latex then
    silence_luatexbase()
end

lwc.callbacks.remove_widows.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 v2.1.0} %%version

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

\catcode`@=11

\input ltluatex % \LuaTeX{}Base

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

\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.
\expandglyphsinfont\the\font 20 20 5
\adjustspacing=2

% Enable \lwc/ by default when the package is loaded.
\lwc@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\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
    \suppressoutererror=1
    \lwcdisablecmd{\beginsection} % Sectioning
\endgroup

% Make the commands public
\let\lwcenable=\lwc@enable
\let\lwcdisable=\lwc@disable
\let\lwcdebug=\lwc@debug
```

```
\let\iflwc=\lwc@if@enabled
\let\lwcnobreak=\lwc@nobreak

\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{v2.1.0}{2022-05-14} %%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}
    {2022/05/14} %%dashdate
    {v2.1.0} %%version
    {Use Lua to remove widows and orphans}

% Unconditional Package Loads
\RequirePackage { l3keys2e }

% 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 },

    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
\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.
\bool_if:NT \g__lwc_use_microtype_bool {
    \hook_gput_code:nnn { begindocument / before } { \c__lwc_name_str } {
        \@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
}

% 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
}

%%% 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 },

    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
\exp_args:NV \ProcessKeysPackageOptions { \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
```

t-lua-widow-control.mkxl/mkiv

```
%D \module
%D   [     file=t-lua-widow-control,
%D      version=2.1.0, %%version
%D        title=lua-widow-control,
%D     subtitle=\ConTeXt module for lua-widow-control,
%D       author=Max Chernoff,
%D         date=2022-05-14, %%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

\installnamespace{lwc}

\installcommandhandler \????lwc {lwc} \????lwc

\newdimen\lwc_emergency_stretch
\newcount\lwc_max_cost
\appendtoks
    \lwc_emergency_stretch=\lwcparameter{emergencystretch}

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

    \lwc_debug{\lwcparameter{debug}}

    \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]
\to\everysetuplwc

\ctxloadluafile{lua-widow-control}

\setuplwc[
    emergencystretch=3em,
    \c!state=\v!start,
    debug=\v!stop,
    orphanpenalty=1,
    widowpenalty=1,
    brokenpenalty=1,
    nobreak=keep,
    maxcost=2147483647,
]

% 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%
}

\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 <v2.1.0>} %%version
\_namespace{lwc}

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

\_newdimen\lwcemergencystretch
\lwcemergencystretch=3em

\_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
\_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
```