LUA-WIDOW-CONTROL

Max Chernoff

v 2.1.1

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 *TUGboat* article[1] distributed with this manual (Links: **local**, **CTAN**, **GitHub**). 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.

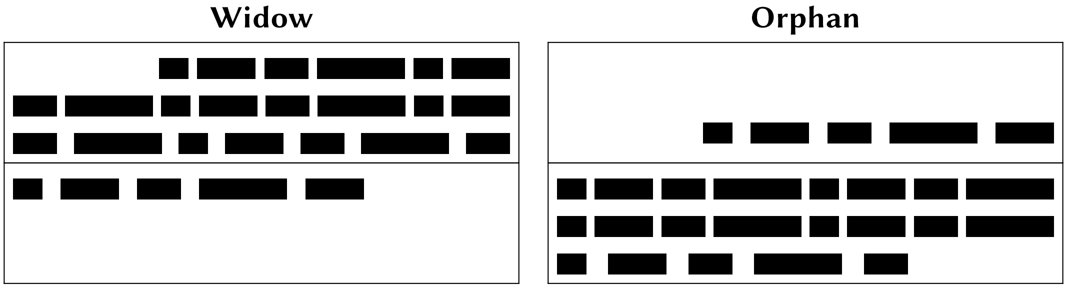


Figure 1 A visual comparison of widows and orphans.

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*[2] ("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.

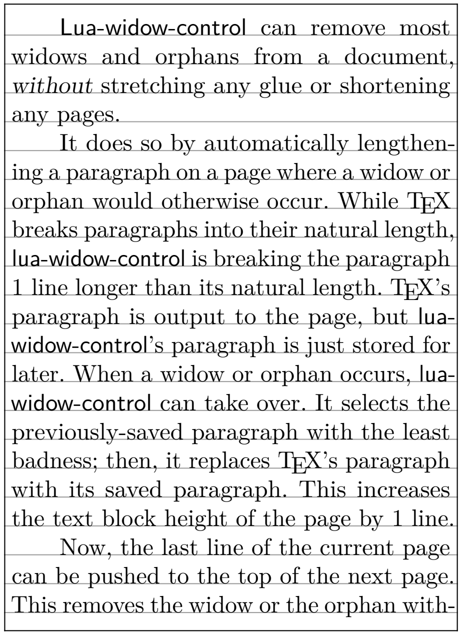
To reduce widows and orphans, TeX sets weakly-positive penalties between the first and second lines of a paragraph to prevent orphans, and between the penultimate and final lines to prevent widows.

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

Ignore



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

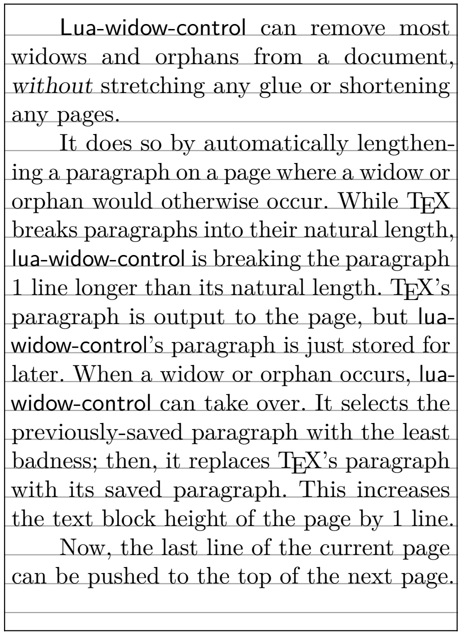
It does so by automatically lengthening a paragraph on a page where a widow or orphan would otherwise occur. While TeX breaks paragraphs into their natural length, lua-widow-control is breaking the paragraph 1 line longer than its natural length. TeX's paragraph is output to the page, but lua-widow-control's 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 without 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 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.

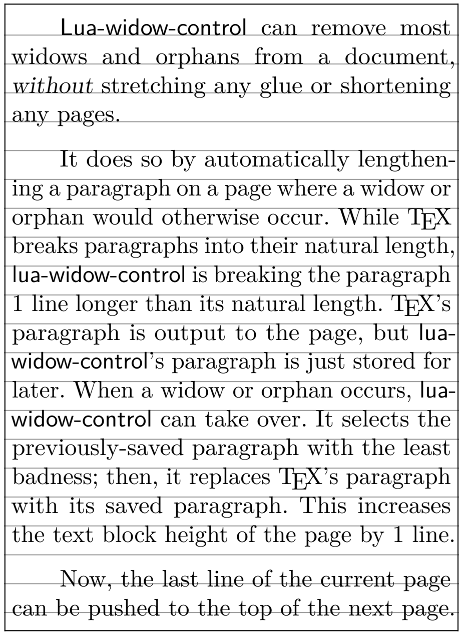
Now, the last line of the current page can be pushed to the top of the next page.

This removes the widow or the orphan without 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 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.

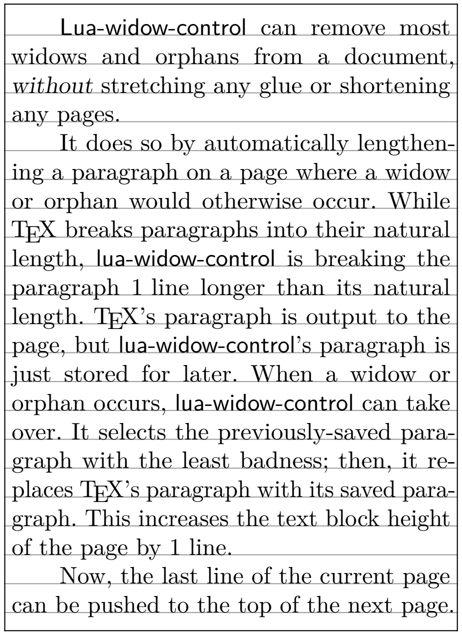
Now, the last line of the current page can be pushed to the top of the next page.

This removes the widow or the orphan without 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 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 without creating any additional work.

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

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

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.

“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 certainly be met by all but the most minimal of TeX installations.

Plain TeX Lua-widow-control requires LuaTeX ($\geq$ 0.85) and the most recent version of luatexbase (2015/10/04). Any version of TeX Live $\geq$ 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.

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 ConTeXtcommands *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.

\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.

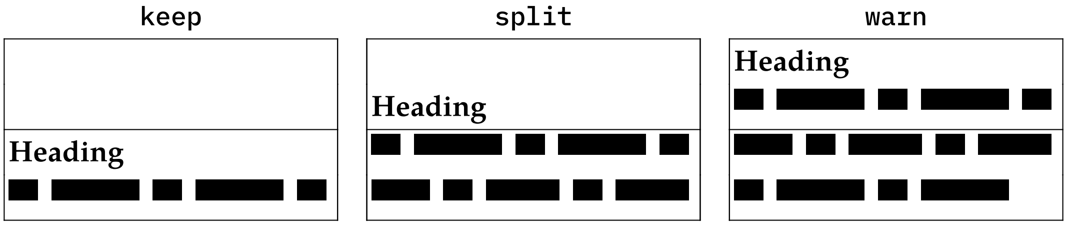


Figure 2 A visual comparison of the various `nobreak` options, where each box represents a different page.

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; however, 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. 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.

Currently, these presets are LaTeX-only.

LaTeX `\lwcsetup{⟨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.

COMPATIBILITY

The lua-widow-control implementation is almost entirely in Lua, with only a minimal TeX footprint. It doesn’t modify the output routine, inserts/floats, `\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.

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.

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.

Simple ConTeXt documents tend to be fine, but many advanced ConTeXt features rely heavily on Lua and can thus be disturbed by lua-widow-control. This is not a huge issue—the lua-widow-control manual is written in ConTeXt—but lua-widow-control is inevitably more reliable with Plain TeX and LaTeX than with ConTeXt.

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

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 [2].

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.

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 wherever reasonable; however, I will rarely make modifications to the algorithm when I can improve the output quality.

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[2]. 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 `0pt plus 0.8\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[3], so this is still an improvement.
- Lua-widow-control only attempts to expand paragraphs; it never attempts to shrink them. See the *TUGboat* article[1] §15.3 for further discussion. **(Issue #33)**
- 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.
- When running under LuaMetaTEX (ConTEXt), the log may contain many lines like "`luatex warning > tex: left parfill skip is gone`". These messages are completely harmless (although admittedly quite annoying). **(Issue #7)**
- 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*[4] would solve this; however, this is both NP-complete[5] and impossible to solve in a single pass. Very rarely would such a system remove widows or orphans that lua-widow-control cannot.
- 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)**

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

IMPLEMENTATION

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 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" or format == "luahbtex" then -- Plain
    plain = true
elseif format:find("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")
```

```
-- Local versions of globals
local abs = math.abs
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
local vpack = node.vpack

-- 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 = "2.1.1", --%%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
```

```lua
        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
```

```lua
--- 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")

    local vsize = tex.dimen.vsize
    local orig_height_diff = vpack(head).height - vsize

    --[[
        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

local new_height_diff = vpack(head_save).height - vsize
-- We need the original height discrepancy in case there are \\vfill's
local net_height_diff = orig_height_diff - new_height_diff

--[[ The final \\box255 needs to be exactly \\vsize tall to avoid
     over/underfull box warnings, so we correct any discrepancies
     here.
  ]]
if abs(net_height_diff) > 0 and
```

```
       -- A difference larger than 0.25\\baselineskip is probably not from \lwc/
       abs(net_height_diff) < tex.skip.baselineskip.width / 4
    then
        local bottom_glue = new_node("glue")
        bottom_glue.width = net_height_diff
        last(head_save).next = bottom_glue
    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

```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 v2.1.1} %%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{v2.1.1}{2022-05-20} %%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
    {v2.1.1} %%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.
\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
}

% 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
```

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:
    2022/05/20 %%dashdate
    v2.1.1 %%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=2.1.1, %%version
%D        title=lua-widow-control,
%D     subtitle=\ConTeXt module for lua-widow-control,
%D       author=Max Chernoff,
%D         date=2022-05-20, %%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 <v2.1.1>} %%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
```