LUA-WIDOW-CONTROL

Max Chernoff

v 1.1.4

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

Lua-widow-control is a Plain TEX/LATEX/ConTEXt 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 **Installation and 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.

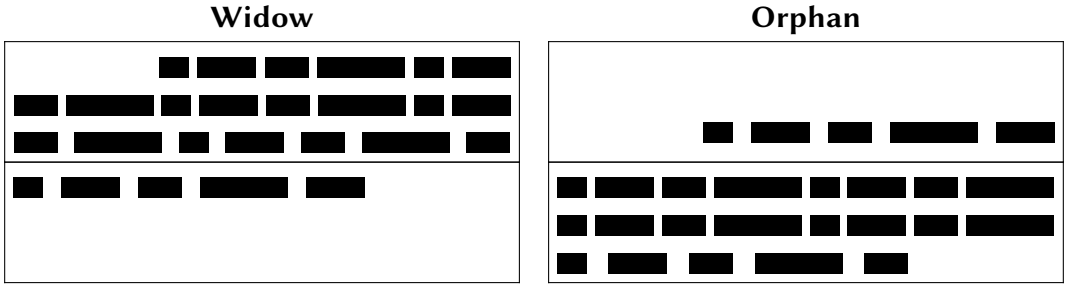

Figure 1 A visual comparison of widows and orphans.

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.

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.

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.

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.

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 (≥ 2015/01/01), microtype (any version), and etoolbox (any version). Any version of TEX Live ≥ 2016 will meet these requirements.

ConTEXt Lua-widow-control requires ConTEXt MKXL (luametaTEX).

LOADING THE PACKAGE

Plain TEX `\input lua-widow-control`

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

ConTEXt `\usemodule[lua-widow-control]`

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.

ADVANCED USAGE

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 does provide a few commands.

Plain TEX

Enable (default)	`\lwcenable`
Disable	`\lwcdisable`
`\emergencystretch`	`\lwcemergencystretch=3em`
Selectively Disable	`\lwcdisablecmd{\cmd}`

LATEX

Enable (default)	`\lwcenable`
Disable	`\lwcdisable`
`\emergencystretch`	`\setlength{\lwcemergencystretch}{3em}`
Selectively Disable	`\lwcdisablecmd{\cmd}`

ConTEXt

Enable (default)	`\setuplwc[state = start]`
Disable	`\setuplwc[state = stop]`
`\emergencystretch`	`\setuplwc[emergencystretch = 3em]`
Selectively Disable	`\prependtoks\lwc@patch@pre \to\everybeforefoo`
	`\prependtoks\lwc@patch@post\to\everyafterfoo`

`\emergencystretch`

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

Lua-widow-control will only use 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 its pretty rare that a paragraph that requires any `\emergencystretch` will actually be used on the page.

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. Place `\lwcdisablecmd{\cmd}` in the preamble, and lua-widow-control will not expand any arguments of `\cmd` in the document.

Lua-widow-control automatically patches the default LATEX, ConTEXt, and Plain TEX section commands, so you shouldn't need to patch these yourself; however, lua-widow-control does **not** patch the non-standard section commands provided by memoir, KOMA-script, titlesec, and others. You'll need to patch these yourself.

Under ConTEXt, you need to use a different method to selectively disable lua-widow-control. Instead of using `\lwcdisablecmd`, you should use one of the preexisting `\everyfoo` hooks as shown in the table above since patching commands is rarely advisable in ConTEXt

KNOWN ISSUES

- Lua-widow-control will not expand the first paragraph of a document.
- 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.

- 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.
- Lua-widow-control will rarely raise a "`Circular node list detected!`" warning. This occurs when the replacement paragraph node list loops back on itself. Since there is no "end" to the paragraph, lua-widow-control cannot splice the paragraph into the page. The only reasonable option in this scenario is to stop processing the page, without removing the widow or orphan.

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 demerits. 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-widow-control
    https://github.com/gucci-on-fleek/lua-widow-control
    SPDX-License-Identifier: MPL-2.0+
    SPDX-FileCopyrightText: 2021 Max Chernoff
  ]]

lwc = {}
lwc.name = "lua-widow-control"

--[[
    lua-widow-control is intended to be format-agonistic. It only runs on LuaTeX,
    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

if format:find('cont') then -- cont-en, cont-fr, cont-nl, ...
    lwc.context = true
elseif format:find('latex') then -- lualatex, lualatex-dev, ...
    lwc.latex = true
elseif format == 'luatex' then -- Plain
    lwc.plain = 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.
  ]]
local last = node.slide
local copy = node.copy_list
local par_id = node.id("par")
local glue_id = node.id("glue")
local set_attribute = node.set_attribute
local has_attribute = node.has_attribute
local flush_list = node.flush_list or node.flushlist
local free = node.free
local min_col_width = tex.sp("25em")
local maxdimen = 1073741823 -- \maxdimen in sp
```

```
--[[
    This error is raised in the following circumstances:
      - When the user manually loads the Lua module without loading LuaTeXBase
      - When the package is used with an unsupported format
    Both of these are pretty unlikely, but it can't hurt to check.
  ]]
assert(lwc.context or luatexbase, [[

    This module requires a supported callback library. Please
    follow the following format-dependant instructions:
      - LaTeX: Use a version built after 2015-01-01, or include
             `\usepackage{luatexbase}' before loading this module.
      - Plain: Include `\input ltluatex' before loading this module.
      - ConTeXt: Use the LMTX version.
]])

--[[
    Package/module initialization
  ]]
if lwc.context then
    lwc.warning = logs.reporter("module", lwc.name)
    lwc.attribute = attributes.public(lwc.name)
    lwc.contrib_head = 'contributehead' -- For luametaTeX
    lwc.stretch_order = "stretchorder"
elseif lwc.plain or lwc.latex then
    luatexbase.provides_module {
        name = lwc.name,
        date = "2022/02/04", --%%date
        version = "1.1.4", --%%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.
        ]],
    }
    lwc.warning = function(str) luatexbase.module_warning(lwc.name, str) end
    lwc.attribute = luatexbase.new_attribute(lwc.name)
    lwc.contrib_head = 'contrib_head' -- For LuaTeX
    lwc.stretch_order = "stretch_order"
else -- uh oh
    error [[
        Unsupported format.
```

```
        Please use (Lua)LaTeX, Plain (Lua)TeX, or ConTeXt (MKXL/LMTX)
    ]]
end

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

if tex.interlinepenalty ~= 0 then
    lwc.warning [[
\interlinepenalty is set to a non-zero value.
This may prevent lua-widow-control from
properly functioning.
]]
end


--[[
    Function definitions
  ]]

--- Create a table of functions to enable or disable a given callback
--- @param t table Parameters of the callback to create
---     callback: str = The LuaTeX callback name
---     func: function = The function to call
---     name: str = The name/ID of the callback
---     category: str = The category for a ConTeXt "Action"
---     position: str = The "position" for a ConTeXt "Action"
---     lowlevel: bool = 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
function lwc.register_callback(t)
    if lwc.plain or lwc.latex then -- Both use LuaTeXBase 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 lwc.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 lwc.context and t.lowlevel then
        --[[
            Some of the callbacks in ConTEXt have no associated "actions". Unlike
            with LuaTEXbase, 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
            lua-widow-control 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,
        }
    end
end


--- Saves each paragraph, but lengthened by 1 line
function lwc.save_paragraphs(head)
    -- Prevent the "underfull hbox" warnings when we store a potential paragraph
    lwc.callbacks.disable_box_warnings.enable()

    -- Ensure that we were actually given a par (only under ConTEXt for some reason)
    if head.id ~= par_id and lwc.context then
        return head
    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 (TEXBook p. 104), except with narrow columns
    local parfillskip = last(new_head)
    if parfillskip.id == glue_id and tex.hsize > min_col_width then
            parfillskip[lwc.stretch_order] = 0
```

```
            parfillskip.stretch = 0.8 * tex.hsize -- Last line must be at least 20% long
    end

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

    -- Break the natural paragraph so we know how long it was
    local natural_node, natural_info = tex.linebreak(copy(head))
    flush_list(natural_node)

    lwc.callbacks.disable_box_warnings.disable()

    -- If we can't lengthen the paragraph, assign a very large demerit value
    local long_demerits
    if long_info.prevgraf == natural_info.prevgraf then
        long_demerits = maxdimen
    else
        long_demerits = long_info.demerits
    end

    -- Offset the accumulated \prevdepth
    local prevdepth = node.new("glue")
    prevdepth.width = natural_info.prevdepth - long_info.prevdepth
    last(long_node).next = prevdepth

    table.insert(lwc.paragraphs, {demerits = long_demerits, node = long_node})

    -- luametaTEX 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 lua-widow-control, and the value corresponds to the
--- paragraphs index, which is negated for the end of the paragraph.
function lwc.mark_paragraphs(head)
    set_attribute(head, lwc.attribute, #lwc.paragraphs)
    set_attribute(last(head), lwc.attribute, -1 * #lwc.paragraphs)
```

```
    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 returns
--- `nil` if the list is circular.
local function safe_last(head)
    local ids = {}

    while head.next do
        local id = node.is_node(head) -- Returns the internal node id

        if ids[id] then
            lwc.warning("Circular node list detected!")
            return nil
        end

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

    return head
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.
function lwc.remove_widows(head)
    local penalty = tex.outputpenalty
    local paragraphs = lwc.paragraphs

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

```
if  penalty    >=  10000 or
    penalty    <= -10000 or
    penalty    ==      0 or
   #paragraphs ==      0 then
        return head
end

local head_save = head -- Save the start of the `head` linked-list

--[[
    Find the paragraph on the page with the minimum penalty.

    This would be a 1-liner in Python or JavaScript, but Lua is pretty low-level,
    so there's quite a bit of code here.
  ]]
local paragraph_index = 1
local minimum_demerits = paragraphs[paragraph_index].demerits

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

local target_node = paragraphs[paragraph_index].node
local clear_flag = false

-- Loop through all of the nodes on the page
while head do
    -- Insert the start of the replacement paragraph
    if has_attribute(head, lwc.attribute, paragraph_index) then
        local current = head
        head.prev.next = target_node

        -- Abort if we've created a loop
        if not safe_last(target_node) then
            head.prev.next = current
```

```
                lwc.warning("Widow/Orphan NOT removed!")
                break
            end

            clear_flag = true
        end

        -- Insert the end of the replacement paragraph
        if has_attribute(head, lwc.attribute, -1 * paragraph_index) then
            last(target_node).next = head.next
            clear_flag = false
        end

        -- Start of final paragraph
        if has_attribute(head, lwc.attribute, #paragraphs) then
            local last_line = copy(last(head))

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

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

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

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

    return head_save
end


-- Add all of the callbacks
lwc.callbacks = {
    disable_box_warnings = lwc.register_callback({
        callback = "hpack_quality",
        func     = function() end,
        name     = "disable_box_warnings",
        lowlevel = true,
```

```
    }),
    remove_widows = lwc.register_callback({
        callback = "pre_output_filter",
        func     = lwc.remove_widows,
        name     = "remove_widows",
        lowlevel = true,
    }),
    save_paragraphs = lwc.register_callback({
        callback = "pre_linebreak_filter",
        func     = lwc.save_paragraphs,
        name     = "save_paragraphs",
        category = "processors",
        position = "after",
    }),
    mark_paragraphs = lwc.register_callback({
        callback = "post_linebreak_filter",
        func     = lwc.mark_paragraphs,
        name     = "mark_paragraphs",
        category = "finalizers",
        position = "after",
    }),
}


local enabled = false
function lwc.enable_callbacks()
    if not enabled then
        lwc.callbacks.remove_widows.enable()
        lwc.callbacks.save_paragraphs.enable()
        lwc.callbacks.mark_paragraphs.enable()

        enabled = true
    else
        lwc.warning("Already enabled")
    end
end


function lwc.disable_callbacks()
    if enabled then
        lwc.callbacks.save_paragraphs.disable()
        lwc.callbacks.mark_paragraphs.disable()
        --[[
            We do not disable `remove_widows` callback, since we still want
```

```
            to expand any of the previously-saved paragraphs if we hit an orphan
            or a widow.
          ]]

        enabled = false
    else
        lwc.warning("Already disabled")
    end
end

function lwc.if_lwc_enabled()
    if enabled then
        tex.sprint("\\iftrue")
    else
        tex.sprint("\\iffalse")
    end
end


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: 2021 Max Chernoff

\wlog{lua-widow-control v1.1.4} %%version

\input ltluatex % LuaTeXBase

\clubpenalty=1
\widowpenalty=1
\displaywidowpenalty=0
\interlinepenalty=0
\brokenpenalty=0

\newdimen\lwcemergencystretch
\lwcemergencystretch=3em

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

% Here, we enable font expansion/contraction. It isn't strictly necessary for
% lua-widow-control's functionality; however, it is required for the
```

```
% lengthened paragraphs to not have terrible spacing.
\expandglyphsinfont\the\font 20 20 5
\adjustspacing=2

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

% Enable lua-widow-control 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 lua-widow-control for certain commands.
\catcode`@=11

% We should only reenable lua-widow-control at the end if it was already enabled.
\newif\iflwc@should@reenable

\def\lwc@patch@pre{\iflwc%
    \lwc@should@reenabletrue%
    \lwcdisable%
\fi}

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

\def\lwcdisablecmd#1{%
    \ifdefined#1
        \expandafter\def\expandafter#1\expandafter{\lwc@patch@pre #1\lwc@patch@post}
    \fi
}
\catcode`@=12

\begingroup
    \suppressoutererror=1
    \lwcdisablecmd{\beginsection} % Sectioning
\endgroup

\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: 2021 Max Chernoff

\NeedsTeXFormat{LaTeX2e}[2015/01/01] % Formats built after 2015 include LuaTeXBase
\ProvidesPackage{lua-widow-control}%
    [2022/02/04 v1.1.4] %%version %%date

\clubpenalty=1
\widowpenalty=1
\displaywidowpenalty=0
\interlinepenalty=0
\brokenpenalty=0

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

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

% Here, we enable font expansion/contraction. It isn't strictly necessary for
% lua-widow-control's functionality; however, it is required for the
% lengthened paragraphs to not have terrible spacing.
\RequirePackage[final]{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 lua-widow-control 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 lua-widow-control for certain commands.
\RequirePackage{etoolbox}

\newcommand{\lwc@patch@warning}[1]{\PackageWarning{lua-widow-control}{%
    Patching the \protect#1 command failed%
}}

% We should only reenable lua-widow-control at the end if it was already enabled.
```

```
\newif\iflwc@should@reenable

\newcommand{\lwc@patch@pre}{\iflwc%
    \lwc@should@reenabletrue%
    \lwcdisable%
\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=1.1.4, %%version
%D        title=lua-widow-control,
%D     subtitle=ConTEXt module for lua-widow-control,
%D       author=Max Chernoff,
%D         date=2022-02-04, %%date
%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\lwcemergencystretch
\appendtoks
    \lwcemergencystretch=\lwcparameter{emergencystretch}
```

```
\to\everysetuplwc

\appendtoks
    \doifelse{\lwcparameter{\c!state}}\v!start{
        \ctxlua{lwc.enable_callbacks()}
    }{
        \ctxlua{lwc.disable_callbacks()}
    }
\to\everysetuplwc

\define\iflwc{\ctxlua{lwc.if_lwc_enabled()}}

\ctxloadluafile{lua-widow-control}

\setuplwc[emergencystretch=3em, \c!state=\v!start]

% We can't just set the penalties because they will be reset automatically
% at \starttext.
\startsetups[*default]
    \clubpenalty=1
    \widowpenalty=1
    \displaywidowpenalty=0
    \interlinepenalty=0
    \brokenpenalty=0
\stopsetups

\setups[*default]

% Here, we enable font expansion/contraction. It isn't strictly necessary for
% lua-widow-control'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 lua-widow-control for certain commands.
% We should only reenable lua-widow-control at the end if it was already enabled.
\newif\iflwc@should@reenable

\define\lwc@patch@pre{\iflwc%
    \lwc@should@reenabletrue%
    \setuplwc[state=stop]%
\fi}
```

```
\define\lwc@patch@post{\iflwc@should@reenable%
    \setuplwc[state=start]%
\fi}

\prependtoks\lwc@patch@pre\to\everybeforesectionheadhandle % Sectioning
\prependtoks\lwc@patch@post\to\everyaftersectionheadhandle

\protect
\stopmodule
```

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]

\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
```