wkgtk-html2pdf c++ API manual

rev V0.0.13.20260212_01

Index

1. Introduction

wkgtk-html2pdf is a powerful and easy-to-use API for converting HTML content into high-quality PDF documents using WebKitGTK. Developed as a modern alternative to legacy tools like wkhtmltopdf—which has been archived since 2023—this project fills the gap for reliable, up-to-date HTML-to-PDF conversion. It provides a clean, intuitive C++ API that simplifies HTML generation by eliminating the need for string literals and explicit closing tags, making it ideal for embedding in applications. The tool supports advanced features such as internal anchors, links, and nested sidebar indexing, enabling the creation of structured, navigable PDFs. A simple command-line interface is also available for quick, one-off conversions. With its focus on maintainability and modern development practices, wkgtk-html2pdf is designed to be a reliable choice for developers seeking a dependable solution for PDF generation.

This project is entirely inspired by wkhtmltopdf as when we were searching for a replacement we simply could not find one, and even the paid for options seemed overly complex where they needn't be and severely lacking where they should. We have tried to iron out some of the issues we have encountered along the way and have produced this manual to assist with those unavoidable issues that must be worked around. We hope that you find this project useful and we would be very much appreciative of any contribution, be that funding, documentation, testing, or design.

wkgtk-html2pdf includes a set of built-in, pre-configured CSS templates for all standard ISO and US paper sizes, ensuring compatibility and consistency across different regions and use cases. These templates are designed to prevent the generation of extraneous blank pages by precisely defining page dimensions, margins, and layout constraints using CSS custom properties.

Each template uses a clean, modular structure with variables for page width, height, and margin, allowing for easy customization while maintaining reliable output. A lightweight JavaScript utility is included to monitor content overflow in real time—when content exceeds the available space, the subpage border turns red; when it fits perfectly, the border turns green. This visual feedback helps you quickly identify and resolve layout issues before conversion.

The templates also include print-specific CSS rules that ensure clean, artefact-free output when the HTML is rendered to PDF, while maintaining visual guides during development for easier debugging and refinement.

In **wkgtk-html2pdf** we have endeavoured to create a system that does not require you to learn additional languages; all you should need is a reasonable understanding of HTML.

1.1 Reading this manual

This manual is a self fulfilling prophecy as it is developed entirely using wkgtk-html2pdf so the layout the code that is used to write it is in itself a tutorial. If you are uncertain about how something is done then is worth reviewing the HTML and CSS used to generate it.

2. Command line interface (CLI)

The CLI is the simplest way to get started with wkgtk-html2pdf. It allows you to convert HTML files into PDFs with minimal setup.

2.1 Basic Usage

To generate a PDF using the default settings use the following command:

```
html2pdf -i input.html -o output.pdf
```

2.2 Command line options

The following options are available for the cli.

Option	Description
-h, --help	Display the help message with all available options
-v, --verbose	Set the log level (1-7), with higher values providing more detailed output. Logs are written to the system journal
-i, --infile	Specify the source HTML file to convert
-o, --outfile	Specify the output PDF file name
-O, --orientation	Set page orientation: portrait or landscape
-s, --size	Set the page size (Default A4)
	- ISO (A): A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10
	- ISO (B): B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10
	- ISO (C): C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10
	- US: Letter, Legal, Tabloid
	- ANSI: ANSIA, ANSIB, ANSIC, ANSID, ANSIE
	- Architectural: ArchA, ArchB, ArchC, ArchD, ArchE
	- Other: SRA0, SRA1, SRA2, SRA3, SRA4
--index	Create anchor points for indexing: `classic` or `enhanced` for nested sidebar indexing

3. Optimising HTML

To generate a clean, professional PDF, you must optimize your HTML to ensure it aligns with the desired page size and layout. The output is entirely governed by the CSS and HTML structure, so the quality of the final PDF depends on how well your content is designed.

3.1 Key Principles

3.1.1 Use the correct stylesheet

1. Each stylesheet is named according to the page size and orientation it supports (e.g., A4-portrait.css, ANSIA-landscape.css).
2. Link to the appropriate stylesheet in your HTML:

Example:

```
<link rel="stylesheet" href="/usr/share/wk2gtkpdf/A4-portrait.css">
```

Applying the Style Sheet for A4 portrait.

3.1.2 Use the correct classes

1. Use the **.page** class to define the overall page
2. Use the **.subpage** class to define content areas.

Example:

```
<div class="page">
    <div class="subpage">
        <!-- Your page content here -->
    </div>
</div>
```

Initialising page boundaries

3.1.3 Monitor the overflow

1. Include the JavaScript utility to monitor content overflow.
2. If overflow is detected the margin turns red.

This script provides real-time feedback, helping you identify and fix layout issues before conversion.

```
<script src="/usr/share/wk2gtkpdf/overflow-monitor.js><script>
```

Declaring the JavaScript to monitor the overflow.

3.1.4 Avoid common issues

By designing your HTML and CSS carefully, you have full control over the final PDF output. Avoid common issues: Mismatched stylesheets, incorrect classes, or content overflow can lead to blank pages, cut-off content, or incorrect scaling.

The overflow detection script helps you identify and fix layout issues before conversion.

3.1.5 Best Practices

1. **Always test the layout** - Use the overflow detection script to ensure content fits within the page. Adjust the CSS or content as needed to avoid overflow.
2. **Use the correct stylesheet** - Ensure the stylesheet matches the CLI arguments (page size and orientation).
3. **Keep it consistent** - Use the same page size and orientation in both the CSS and CLI arguments.

3.2 Quick start - A4 Portrait Layout

Below is a minimal example that will create a single page PDF and utilise a compact built in JavaScript function to monitor and warn of overflow and missing fonts.

Example:

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="/usr/share/wk2gtkpdf/A4-portrait.css">
</head>
<body>
    <div class="page">
        <div class="subpage">
            <h1>My Document</h1>
            <p>This content will be rendered in A4 portrait format.</p>
        </div>
    </div>
</body>
<script src="/usr/share/wk2gtkpdf/overflow-monitor.js><script>
</html>
```

Minimal single page example.

Each declaration of a **.page** and **.subpage** container will create a new page. Blank pages can also be inserted by declaring a **.page** and **.subpage** container.

4. Anchors

When generating PDFs with wkgtk-html2pdf, you can choose between two indexing modes: classic and enhanced. The difference lies in which elements are included in the index.

4.1 Classic model

In classic mode, all internal links (**<a>** tags with href starting with **#**) are automatically included in the index, regardless of their container or structure.

Example:

```
<li><a href="#reference">Reference</a></li>
<p><a href="#section1">Section 1</a></p>
<div><a href="#appendix">Appendix</a></div>
```

All these anchors are valid.

4.2 Enhanced mode

Enhanced mode (--index enhanced) is an advanced feature that allows you to selectively control which links appear in the PDF index. It only indexes elements with the **class="index-item"** class, giving you fine-grained control over the index.

Example:

```
<!-- This will be included -->
<li class="index-item">
    <a href="#reference">
        <span>Reference</span>
        <span>A1</span>
    </a>
</li>

<!-- This will be ignored -->
<p><a href="#section1">Section 1</a></p>

<!-- This will be included (container type doesn't matter) -->
<div class="index-item">
    <a href="#appendix">Appendix</a>
</div>
```

Enhanced anchor declarations.

In this example only the first and third elements will be included in the index. While this feature may not be essential for most users, it could be useful in specific scenarios where you want to:

1. Exclude certain internal links from the index.
2. Create a more structured or semantic index.

The benefits of enhanced mode will become more clear after you read the section on sidebar indexing below.

Clickable areas marked in grey.

Save for the styling the difference is

Enhanced:

```
<li class="index-item"> <a href="#ref"> <span> Enhanced mode </span> <span> 4.1
</span> </a> </li>
```

A typical enhanced mode link.

Classic:

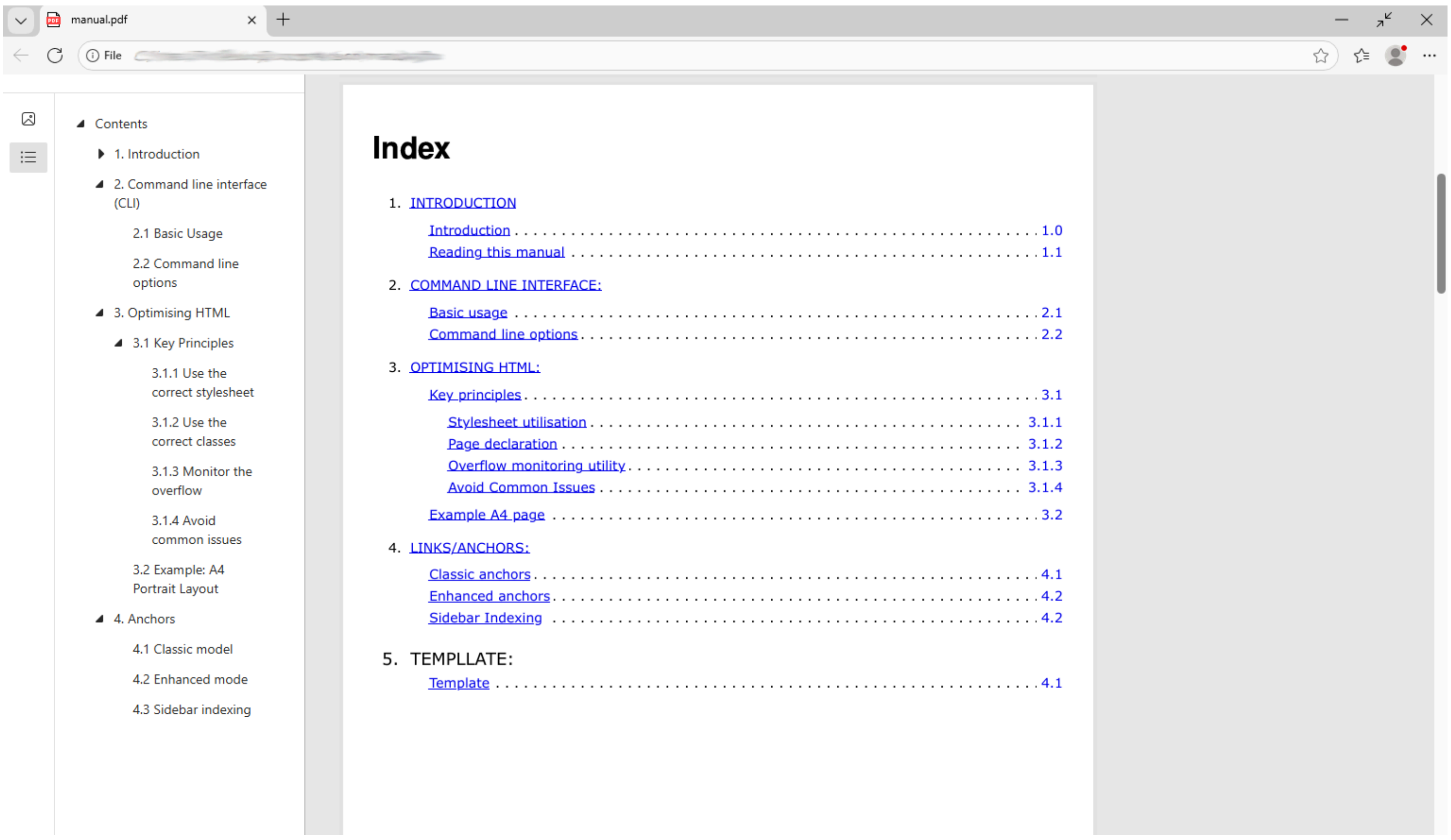
```
<li> <a href="#ref"> <span> Enhanced mode </span> <span> 4.1 </span> </a> </li>
```

A typical demo_classic mode link.

As you can see from the code, theoretically the link should work across the entire line, even in classic mode but it doesn't (even in HTML). Ordinarily when you convert your link to a you lose right right hand side part of the anchor unless you enable enhanced mode

4.3 Sidebar indexing

Most users familiar with PDF manuals will make heavy use of the sidebar index built into the PDF reader as it is often more advantageous than a traditional index due it remaining on scree at all times. With properly formatted HTML and standard indexing practices wkgtk-html2pdf will build a properly nested index for you regardless as to whether you choose to use **static** or **classic** mode.


manual.pdf

Index

Sample pdf with sidebar index.

While it is conventional to have a traditional index page somewhere towards the beginning or end of the document it is not strictly necessary however for an item to appear in the index it must meet the following criteria:

1. The object to be indexed must have an anchor that references it by a unique id
   e.g. <a href=#mylink>
2. The target must have contain a number. If the item is to be nested then it must be separated by a period (see the structure of this manual for an example).

Example:

```
<a href=#myanchor>Take me to your linker</a>
<p id=myanchor>1.1 I am here</p>
```

Creating an index item.

Below is a live example which may or may not appear in the index depending on whether you create the PDF using enhanced index mode or classic.

Take me to your linker

1.1 I am here

A live example in **classic** indexing mode.

The link has been explicitly disabled when the manual is compiled with indexing in **enhanced** mode to further demonstrate why this might be useful. You will note that if you compile this manual in **classic** mode you will actually see the link in the sidebar nested in section 1. Were we not demonstrating how this works that behaviour would be undesirable but we can disable it in advanced mode without causing problems for the rest of the document.

If you wanted to enable the link in enhanced mode then it is as simple as wrapping the anchor in an element with the a class of **"index-item"**.

```
<span class="index-item"><a href="#enhanced-anchor" >Take me to your linker</a>
</span>;
<p id="enhanced-anchor">4.3.1 Enhanced indexing anchor demo</p>
```

Enabling the link in **enhanced** indexing mode.

Enhanced mode links work in both classic and enhanced mode.

Take me to your linker

4.3.1 Enhanced indexing anchor demo

A live example in **enhanced** indexing mode.

The above link will appear in the index. This time you will note that we have explicitly numbered the link **4.3.1** so that it will appear nested in section **4.3** within the index.

4.4 Index page

If you do have an index page then attaching it to the sidebar is as simple as declaring **toc** within the page element.

Example:

```
<div class="page toc">
    <div class="subpage">
        <!-- Your page content here -->
    </div>
</div>
```

A page declared as the commencement of the table of contents.

The table of contents should be declared exactly once per document.

NOTE: If it is declared more than once the the first declaration that wkgtk-html2pdf finds will be used and the rest will be ignored.

When you declare a Table of Contents/Index page it will be attached to the top level **"Contents"** bookmark within the index.

5. Artefacts

5.1 Images

When working with images while developing pages to generate PDF's from the command line interface you should always use full paths or remote links to the image (relative paths will not work).

A randomised image from the internet.

As this is a semi interactive manual if you are viewing this in html you should see a random image generated. This image will be embedded in the PDF upon generation.

Static paths also work:

An image from a static path.

Embedded base64 encoded images also work (the image above is base64 encoded due to the fact that we could not guarantee a static path for this manual but rest assured that static paths work fine as long as they are full rather than relative).

5.2 Fonts

Particular care should be taken when using fonts as if they are not installed on the system that is being used to generate the PDF overflow an unexpected results can occur; particularly if the fallback font is significantly different from the desired font.

Within our suite we have included a JavaScript snippet that if enabled should detect missing fonts and warn you before you generate your PDF. As long as **wkgtk-html2pdf** can see the font it should render the page correctly.

6. Page design helper

When converting or generating HTML that works seamlessly as a PDF there are several issues that may lead to unexpected results. While not an exhaustive list the following are most common:

1. **Unexpected extra pages** - It is not unusual for developers to become extremely frustrated at the fact that they have an extra blank page at the end of their document even though they have done everything right.
2. **Overflow** - Content is often missing or going off the page.

In an attempt to combat these problems that plague HTML to PDF conversion we have incorporated a small JavaScript helper that works alongside our CSS stylesheets to try help combat these problems.

6.1 CSS Style sheets

When you include the appropriate template for your desired page size and declare a **.page** and **.subpage** <div> element. You will note that you now have something on screen that represents the page size of the PDF you are developing. This will include a margin outlined in blue.

All of the page templates are located in

```
/usr/share/wk2gtkpdf/templates
```

Default path of the wkgtk-html2pdf stylesheets.

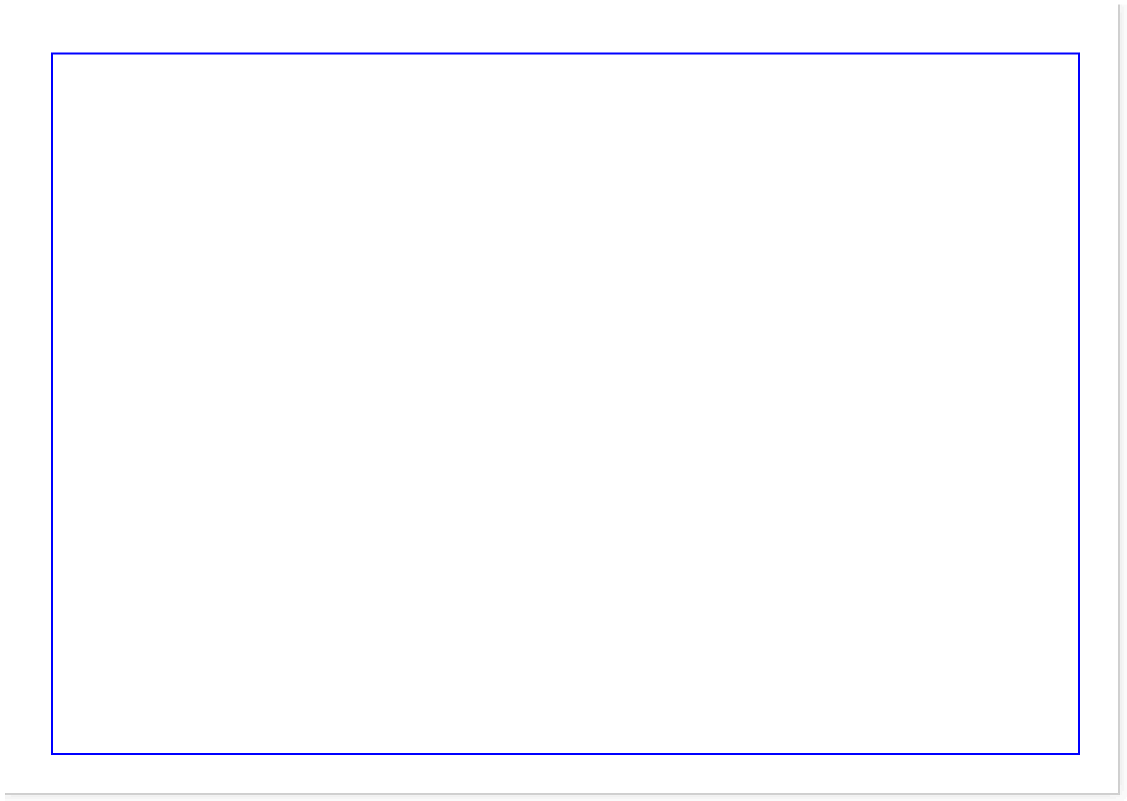
If you wish to change one of the page margins then it is simply a matter of editing the appropriate stylesheet by changing **--page-margin** value as needed.

NOTE: The margin is measured in millimetres(mm).

```
:root {
    --page-width: 210;
    --page-height: 297;
    --page-margin: 8;
}
```

The page margin set in the CSS.

These stylesheets contain calculations that ensure there are no extra blank pages when the PDF is generated. They also provide you with a visual representation of a page including its margins.



A blank A6 page with margins outlined.

As long as you keep your content within the subpage element it should stay within the margins and on the provision that the font is accessible and given the appropriate arguments it should render identically with the command line interface.

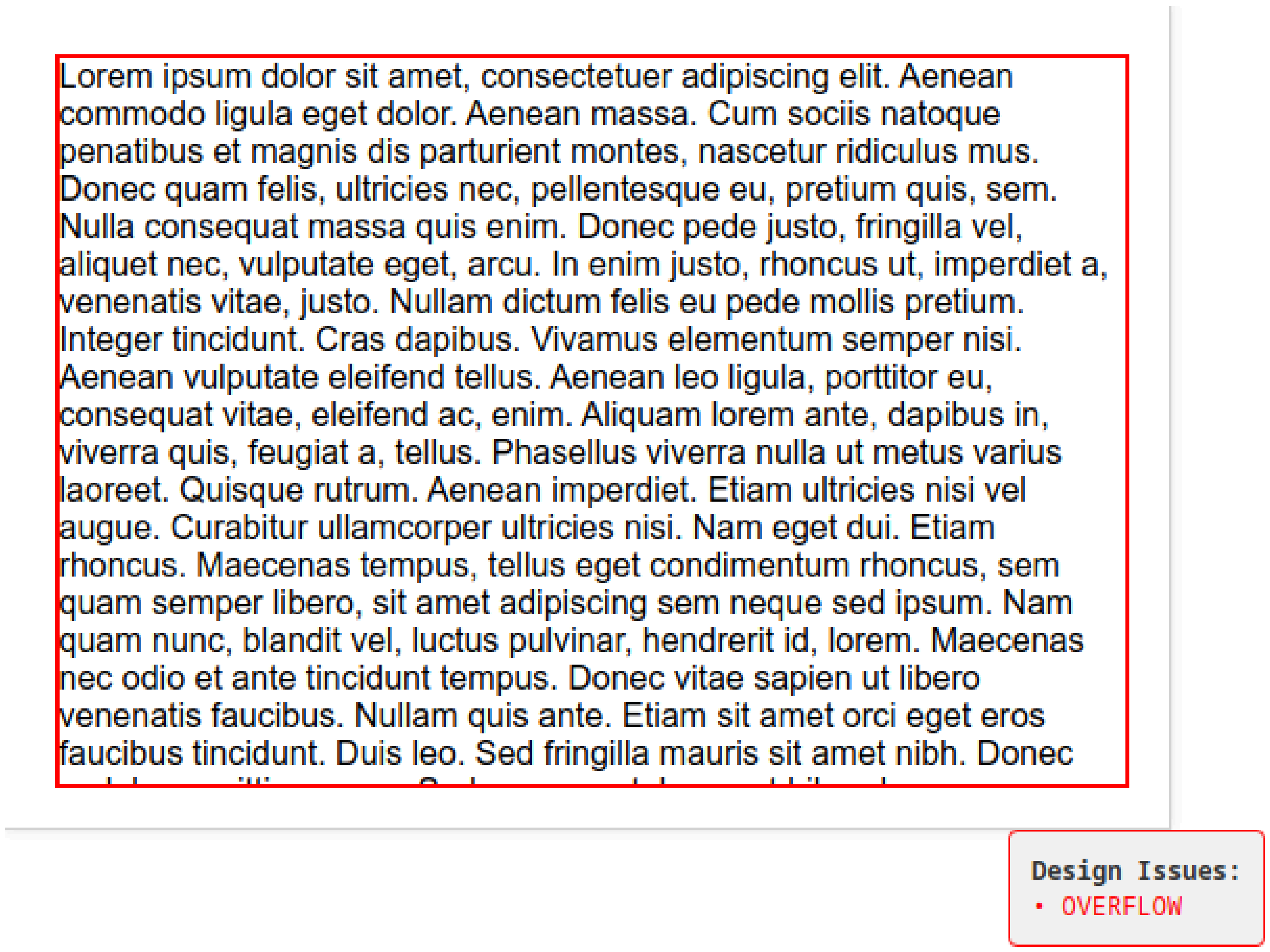
6.2 JavaScript helper utility

As aluded to several times throughout this manual wk2gtk-html2pdf includes a helper utility to aid with design. It is located in the same folder as the CSS templates and can be used to identify problems before trying to generate a PDF.

To use the utility you need to declare it:

```
<script src=""/usr/share/wk2gtkpdf/overflow-monitor.js"><script>
```

Once declared it will automatically monitor changes in the background and warn of any detected Issues in a panel which will appear towards the bottom left corner of the your preview.



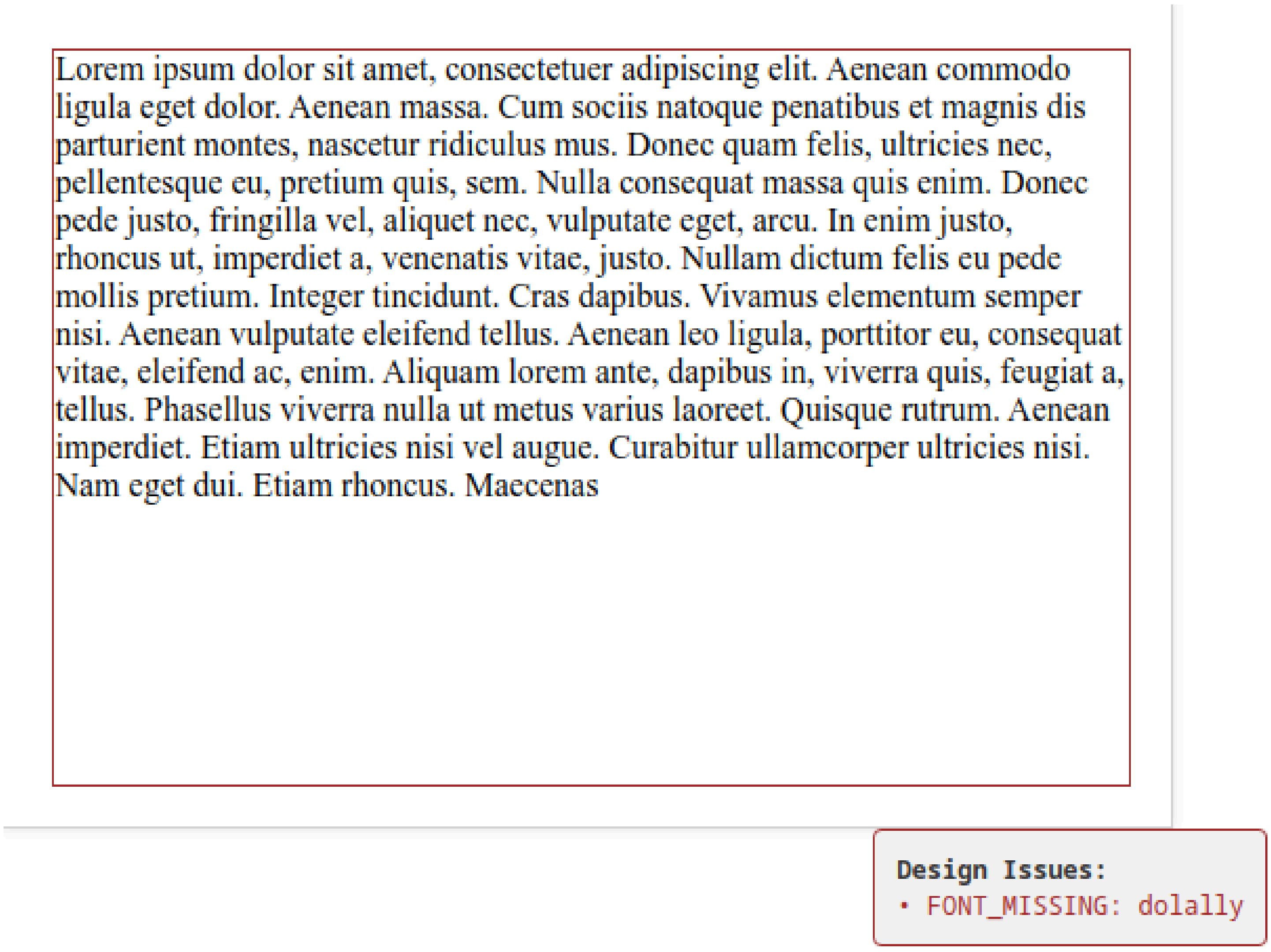
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvinar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus. Donec vitae sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget eros faucibus tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec

```
Design Issues:
• OVERFLOW
```

Notification when the content goes outside of the page margin

In addition to the detection of overflow it should also be capable of detecting missing fonts. This is to be considered advisory and if there are rendering issues the first thing that should be checked is the system wide availability of fonts.

NOTE: Font detection has been checked in most cases however it should be noted that due the manner in which detection is achieved is should not be considered reliable in 100 percent of cases.


Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas

```
Design Issues:
• FONT_MISSING: dolally
```

Notification when a missing font is detected

7. Hints and Tips

7.1 Page Numbering

You will note that the pages within this document are all numbered. That is an automatic feature that we do using CSS and it is done utilising CSS' built in **counter** function:

```
div.subpage:after {
     content: " Page - " counter(page);
     position: absolute;
     bottom: 0px;
     right: 0px;
     z-index: 999;
     padding: 2px 8px;
     border-right: 2px solid #23b8e7;
     font-size: 12px;
 }
```

An automated page counter positioned towards the bottom right of the subpage

Apart from some basic styling this is relatively simple as all that is needed is

```
content: " Page - " counter(page);
```

Page numbering with a prefix

or if you just want a number

```
content: counter(page);
```

Basic page numbering

You will note however that our counter does not start on page 1 (in fact it starts on the 3rd page). To achieve this we need to allow the counter to count up, but hide it on the pages that we do not want it to be visible on, then we need to reset it and make it visible. It is a bit of a hack, but it is easier than keeping track of page numbers manually.

```
/* Hide the counter on pages 1 and 2 */
div.page:first-child div.subpage::after,
div.page:nth-child(2) div.subpage::after {
    display: none;
}
/* Reset the counter on the 3rd page */
div.page:nth-child(3) {
    counter-reset: page;
}
/* Increment the counter every time a new page is encountered */
div.page:not(:first-child) {
    counter-increment: page;
}
```

Restart the counter on page 3

7.2 Remove underlining on PDF

We are all used to seeing linke underlined on web pages but for some it can seem an unnecessary distraction when transcribed to a PDF. Albeit subjective (as all design is) personally I like to get rid of the underline but keep the blue colour just to hint to the user that it is clickable.

To change this or any formatting specifically for pdf generation you use the **@media print** directive in your CSS.

```
@media print {
    a {
        text-decoration-line: none !important;
    }
}
```

Remove the underline from links for PDF

7.3 Remove the visible page margin

Chances are that if you have gone to the trouble of writing a lengthy document in HTML then you may wish to utilise it in HTML format. The problem is that you have a big blue margin around the outside of your page. You may be tempted to set the border to none however that would be ill advised as it could change the formatting. The best thing that you can do to ensure nothing changes is to leave it where it is and just make it white.

```
.subpage {
    border: solid white !important;
}
```

Hide the margin guide

7.4 Development environment setup

Depending on what we are doing we use both of these setups (sometimes even both at once), but as long as you have something to preview and something to write code on you shouldn't run into difficulties.

7.4.1 Pulsar-edit

Without doubt our favorite ide for developing anything HTML is Pulsar-edit. Along with the atom-preview-html plugin you can configure it to perform real time updates as you type, it has plugins for code snippets and when utilised along with our JavaScript plugin can be a real time saver once you get used to it.

```
'.text.html.basic':
    'Page and Subpage':
        'prefix': 'hpage'
        'command': 'insert-page-subpage'
        'body': '<div class="page">\n <div class="subpage">\n </div>\n</div>'
        'description': 'Insert page and subpage container'
    'Code box':
        'prefix': 'bcode'
        'command': 'insert-codebox'
        'body': '<div class="code-container">\n \t<div class="code-container-
sub">\n \t<pre><code>\n </code></pre>\n \t</div>\n</div>'
        'description': 'Insert code snippet container'
```

Our pulsar snippets for inserting the code and page entries to produce this manual

We believe that VS Code is another solid alternative that works similarly however we haven't used it ourselves.

7.4.2 Vim and a browser

If you love vim then you probably don't need us to tell you how to set up your environment but in testing we have found that previewing the output with **Falkon** browser results in the almost real time updates on save without need to manually refresh the page refresh. While probably not the world's greatest browser for daily internet use we have found it the best solution for live previewing while using your favourite text editor (be that vim, nano, or something less hardcore) to design your pages.