WeasyPrint

Release 43

Nov 09, 2018

Contents

1 Documentation contents **3**

1.1 Installing . . . 3
1.2 Tutorial . . . 9
1.3 API . . . 16
1.4 Features . . . 22
1.5 Hacking WeasyPrint . . . 29
1.6 News . . . 32

2 Authors **49**

Python Module Index **51**

WeasyPrint is a visual rendering engine for HTML and CSS that can export to PDF. It aims to support web standards for printing. WeasyPrint is free software made available under a BSD license.

It is based on various libraries but *not* on a full rendering engine like WebKit or Gecko. The CSS layout engine is written in Python, designed for pagination, and meant to be easy to hack on.

- Free software: BSD licensed
- Python 3.4+
- Website: https://weasyprint.org/
- Documentation: https://weasyprint.readthedocs.io/
- Source code and issue tracker: https://github.com/Kozea/WeasyPrint
- Tests: https://travis-ci.org/Kozea/WeasyPrint
- Support: https://www.patreon.com/kozea

CHAPTER 1

Documentation contents

1.1 Installing

WeasyPrint 43 depends on:

- CPython 3.4.0
- cairo 1.15.4[1]
- Pango 1.38.0[2]
- setuptools 30.3.0
- CFFI 0.6
- html5lib 0.999999999
- cairocffi 0.9.0
- tinycss2 0.5
- cssselect2 0.1
- CairoSVG 1.0.20
- Pyphen 0.8
- GDK-PixBuf 2.25.0[3]

Python, cairo, Pango and GDK-PixBuf need to be installed separately. See platform-specific instructions for *Linux*, *macOS* and *Windows* below.

Install WeasyPrint with pip. This will automatically install most of dependencies. You probably need either a virtual environment (venv, recommended) or using `sudo`.

[1] cairo 1.15.4 is best but older versions may work too. The test suite passes on cairo 1.14, and passes with some tests marked as "expected failures" on 1.10 and 1.12 due to behavior changes or bugs in cairo. If you get incomplete SVG renderings, please read #339. If you get invalid PDF files, please read #565. Some PDF metadata including PDF information, hyperlinks and bookmarks require 1.15.4.

[2] pango 1.29.3 is required, but 1.38.0 is needed to handle *@font-face* CSS rules.

[3] Without it, PNG and SVG are the only supported image formats. JPEG, GIF and others are not available.

```
python3 -m venv ./venv
. ./venv/bin/activate
pip install WeasyPrint
```

Now let's try it:

```
weasyprint --help
weasyprint http://weasyprint.org ./weasyprint-website.pdf
```

You should see warnings about unsupported CSS 3 stuff; this is expected. In the PDF you should see the WeasyPrint logo on the first page.

You can also play with *WeasyPrint Navigator* or *WeasyPrint Renderer*. Start it with

```
python -m weasyprint.tools.navigator
```

or

```
python -m weasyprint.tools.renderer
```

and open your browser at http://127.0.0.1:5000/.

If everything goes well, you're ready to *start using* WeasyPrint! Otherwise, please copy the full error message and report the problem.

1.1.1 Linux

Pango, GdkPixbuf, and cairo can not be installed with pip and need to be installed from your platform's packages. CFFI can, but you'd still need their own dependencies. This section lists system packages for CFFI when available, the dependencies otherwise. CFFI needs *libffi* with development files. On Debian, the package is called `libffi-dev`.

If your favorite system is not listed here but you know the package names, tell us so we can add it here.

Debian / Ubuntu

Debian 9.0 Stretch or newer, Ubuntu 16.04 Xenial or newer:

```
sudo apt-get install build-essential python3-dev python3-pip python3-setuptools␣
↪python3-wheel python3-cffi libcairo2 libpango-1.0-0 libpangocairo-1.0-0 libgdk-
↪pixbuf2.0-0 libffi-dev shared-mime-info
```

Fedora

WeasyPrint is packaged for Fedora, but you can install it with pip after installing the following packages:

```
sudo yum install redhat-rpm-config python-devel python-pip python-setuptools python-
↪wheel python-cffi libffi-devel cairo pango gdk-pixbuf2
```

Archlinux

WeasyPrint is available in the AUR, but you can install it with pip after installing the following packages:

```
sudo pacman -S python-pip python-setuptools python-wheel cairo pango gdk-pixbuf2␣
↪libffi pkg-config
```

Gentoo

WeasyPrint is packaged in Gentoo, but you can install it with pip after installing the following packages:

```
emerge pip setuptools wheel cairo pango gdk-pixbuf cffi
```

Alpine

For Alpine Linux 3.6 or newer:

```
apk --update --upgrade add gcc musl-dev jpeg-dev zlib-dev libffi-dev cairo-dev pango-
↪dev gdk-pixbuf
```

1.1.2 macOS

WeasyPrint is automatically installed and tested on virtual macOS machines. The official installation method relies on Homebrew:

```
brew install python3 cairo pango gdk-pixbuf libffi
```

Don't forget to use the *pip3* command to install WeasyPrint, as *pip* may be using the version of Python installed with macOS.

You can also try with Macports, but please notice that this solution is not tested and thus not recommended (**also known as "you're on your own and may end up crying blood with sad dolphins for eternity"**):

```
sudo port install py-pip cairo pango gdk-pixbuf2 libffi
```

1.1.3 Windows

Dear Windows user, please follow these steps carefully.

Really carefully. Don't cheat.

Besides a proper Python installation and a few Python packages, WeasyPrint needs the Pango, Cairo and GDK-PixBuf libraries. They are required for the graphical stuff: Text and image rendering. These libraries aren't Python packages. They are part of GTK+ (formerly known as GIMP Toolkit), and must be installed separately.

The following installation instructions for the GTK+ libraries don't work on Windows XP. That means: Windows Vista or later is required.

Of course you can decide to install ancient WeasyPrint versions with an erstwhile Python, combine it with outdated GTK+ libraries on any Windows version you like, but if you decide to do that **you're on your own, don't even try to report an issue, kittens will die because of you.**

Step 1 - Install Python

Install the latest Python 3.x

- On Windows 32 bit download the "Windows **x86** executable installer"
- On Windows 64 bit download the "Windows **x86-64** executable installer"

Follow the instructions. You may customize your installation as you like, but we suggest that you "Add Python 3.x to PATH" for convenience and let the installer "install pip".

Step 2 - Update pip and setuptools packages

Python is bundled with modules that may have been updated since the release. Please open a *Command Prompt* and execute the following command:

```
python -m pip install --upgrade pip setuptools
```

Step 3 - Install WeasyPrint

In the console window execute the following command to install the WeasyPrint package:

```
python -m pip install WeasyPrint
```

Step 4 - Install the GTK+ libraries

There's one thing you **must** observe:

- If your Python is 32 bit you must use the 32 bit versions of those libraries.
- If your Python is 64 bit you must use the 64 bit versions of those libraries.

If you mismatch the bitness, the warning about kittens dying applies.

In case you forgot which Python you installed, ask Python (in the console window):

```
python --version --version
```

Having installed Python 64 bit you can either use the *GTK+ 64 Bit Installer* or install the 64-bit *GTK+ via MSYS2*.

On Windows 32 bit or if you decided to install Python 32 bit on your Windows 64 bit machine you'll have to install the 32-bit *GTK+ via MSYS2*.

Note: Installing those libraries doesn't mean something extraordinary. It only means that the files must be on your computer and WeasyPrint must be able to find them, which is achieved by putting the path-to-the-libs into your Windows `PATH`.

Install GTK+ with the aid of MSYS2

Sadly the GTK+ Runtime for 32 bit Windows was discontinued in April 2017. Since then developers are advised to either bundle GTK+ with their software (which is beyond the capacities of the WeasyPrint maintainers) or install it through the MSYS2 project.

With the help of MSYS2, both the 32 bit as well as the 64 bit GTK+ can be installed. If you installed the 64 bit Python and don't want to bother with MSYS2, then go ahead and use the *GTK+ 64 Bit Installer*.

MSYS2 is a development environment. We (somehow) mis-use it to only supply the up-to-date GTK+ runtime library files in a subfolder we can inject into our `PATH`. But maybe you get interested in the full powers of MSYS2. It's the perfect tool for experimenting with MinGW and cross-platform development – look at its wiki.

Ok, let's install GTK3+.

- Download and run the MSYS2 installer
  - On 32 bit Windows: "msys2-**i686**-xxxxxxxx.exe"
  - On 64 bit Windows: "msys2-**x86_64**-xxxxxxxx.exe"

  You alternatively may download a zipped archive, unpack it and run `msys2_shell.cmd` as described in the MSYS2 wiki.
- Update the MSYS2 shell with

```
pacman -Syuu
```

  Close the shell by clicking the close button in the upper right corner of the window.
- Restart the MSYS2 shell. Repeat the command

```
pacman -Su
```

  until it says that there are no more packages to update.
- Install the GTK+ package and its dependencies.

  To install the 32 bit (**i686**) GTK run the following command:

```
pacman -S mingw-w64-i686-gtk3
```

  The command for the 64 bit (**x86_64**) version is:

```
pacman -S mingw-w64-x86_64-gtk3
```

  The **x86_64** package cannot be installed in the 32 bit MSYS2!
- Close the shell:

```
exit
```

- Now that all the GTK files needed by WeasyPrint are in the `.\mingw32` respectively in the `.\mingw64` subfolder of your MSYS2 installation directory, we can (and must) make them accessible by injecting the appropriate folder into the `PATH`.

  Let's assume you installed MSYS2 in `C:\msys2`. Then the folder to inject is:
  - `C:\msys2\mingw32\bin` for the 32 bit GTK+
  - `C:\msys2\mingw64\bin` for the 64 bit GTK+

  You can either persist it through *Advanced System Settings* – if you don't know how to do that, read How to set the path and environment variables in Windows – or temporarily inject the folder before you run WeasyPrint.

GTK+ 64 Bit Installer

If your Python is 64 bit you can use an installer extracted from MSYS2 and provided by Tom Schoonjans.

- Download and run the latest gtk3-runtime-x.x.x-x-x-x-ts-win64.exe
- If you prefer to manage your `PATH` environment varaiable yourself you should uncheck "Set up PATH environment variable to include GTK+" and supply it later – either persist it through *Advanced System Settings* or temporarily inject it before you run WeasyPrint.

Note: Checking the option doesn't insert the GTK-path at the beginning of your system `PATH`, but rather **appends** it. If there is alread another (outdated) GTK on your `PATH` this will lead to unpleasant problems.

In any case: When executing WeasyPrint the GTK libraries must be on its `PATH`.

Step 5 - Run WeasyPrint

Now that everything is in place you can test WeasyPrint.

Open a fresh *Command Prompt* and execute

```
python -m weasyprint http://weasyprint.org weasyprint.pdf
```

If you get an error like `OSError: dlopen() failed to load a library: cairo / cairo-2` it's probably because Cairo (or another GTK+ library mentioned in the error message) is not properly available in the folders listed in your `PATH` environment variable.

Since you didn't cheat and followed the instructions the up-to-date and complete set of GTK libraries **must** be present and the error is an error.

Lets find out. Enter the following command:

```
WHERE libcairo-2.dll
```

This should respond with *path\to\recently\installed\gtk\binaries\libcairo-2.dll*, for example:

```
C:\msys2\mingw64\bin\libcairo-2.dll
```

If your system answers with *nothing found* or returns a filename not related to your recently-installed-gtk or lists more than one location and the first file in the list isn't actually in a subfolder of your recently-installed-gtk, then we have caught the culprit.

Depending on the GTK installation route you took, the proper folder name is something along the lines of:

- `C:\msys2\mingw32\bin`
- `C:\msys2\mingw34\bin`
- `C:\Program Files\GTK3-Runtime Win64\bin`

Determine the correct folder and execute the following commands, replace `<path-to-recently-installed-gtk>` accordingly:

```
SET PROPER_GTK_FOLDER=<path-to-recently-installed-gtk>
SET PATH=%PROPER_GTK_FOLDER%;%PATH%
```

This puts the appropriate GTK at the beginning of your `PATH` and it's files are the first found when WeasyPrint requires them.

Call WeasyPrint again:

```
python -m weasyprint http://weasyprint.org weasyprint.pdf.
```

If the error is gone you should either fix your `PATH` permanently (via *Advanced System Settings*) or execute the above `SET PATH` command by default (once!) before you start using WeasyPrint.

If the error still occurs and if you really didn't cheat then you are allowed to open a new issue. You can also find extra help in this bug report. If you cheated, then, you know: Kittens already died.

1.2 Tutorial

1.2.1 As a standalone program

Once you have WeasyPrint *installed*, you should have a `weasyprint` executable. Using it can be as simple as this:

```
weasyprint http://weasyprint.org /tmp/weasyprint-website.pdf
```

You may see warnings on *stderr* about unsupported CSS properties. See *Command-line API* for the details of all available options.

In particular, the `-s` option can add a filename for a *user stylesheet*. For quick experimentation however, you may not want to create a file. In bash or zsh, you can use the shell's redirection instead:

```
weasyprint http://weasyprint.org /tmp/weasyprint-website.pdf \
    -s <(echo 'body { font-family: serif !important }')
```

If you have many documents to convert you may prefer using the Python API in long-lived processes to avoid paying the start-up costs every time.

Adjusting Document Dimensions

Currently, WeasyPrint does not provide support for adjusting page size or document margins via command-line flags. This is best accomplished with the CSS `@page` at-rule. Consider the following example:

```
@page {
  size: Letter; /* Change from the default size of A4 */
  margin: 2.5cm; /* Set margin on each page */
}
```

There is much more which can be achieved with the `@page` at-rule, such as page numbers, headers, etc. Read more about the page at-rule, and find an example here.

1.2.2 As a Python library

Attention: Using WeasyPrint with untrusted HTML or untrusted CSS may lead to various *security problems*.

Quickstart

The Python version of the above example goes like this:

```
from weasyprint import HTML
HTML('http://weasyprint.org/').write_pdf('/tmp/weasyprint-website.pdf')
```

... or with the inline stylesheet:

```
from weasyprint import HTML, CSS
HTML('http://weasyprint.org/').write_pdf('/tmp/weasyprint-website.pdf',
    stylesheets=[CSS(string='body { font-family: serif !important }')])
```

Instantiating HTML and CSS objects

If you have a file name, an absolute URL or a readable file-like object, you can just pass it to *HTML* or *CSS* to create an instance. Alternatively, use a named argument so that no guessing is involved:

```
from weasyprint import HTML

HTML('../foo.html')  # Same as ...
HTML(filename='../foo.html')

HTML('http://weasyprint.org')  # Same as ...
HTML(url='http://weasyprint.org')

HTML(sys.stdin)  # Same as ...
HTML(file_obj=sys.stdin)
```

If you have a byte string or Unicode string already in memory you can also pass that, although the argument must be named:

```
from weasyprint import HTML

# HTML('<h1>foo') would be filename
HTML(string='''
    <h1>The title</h1>
    <p>Content goes here
''')
CSS(string='@page { size: A3; margin: 1cm }')
```

If you have `@font-face` rules in your CSS, you have to create a `FontConfiguration` object:

```
from weasyprint import HTML, CSS
from weasyprint.fonts import FontConfiguration

font_config = FontConfiguration()
html = HTML(string='<h1>The title</h1>')
css = CSS(string='''
    @font-face {
        font-family: Gentium;
        src: url(http://example.com/fonts/Gentium.otf);
    }
    h1 { font-family: Gentium }''', font_config=font_config)
html.write_pdf(
    '/tmp/example.pdf', stylesheets=[css],
    font_config=font_config)
```

Rendering to a single file

Once you have a *HTML* object, call its *write_pdf()* or *write_png()* method to get the rendered document in a single PDF or PNG file.

Without arguments, these methods return a byte string in memory. If you pass a file name or a writable file-like object, they will write there directly instead. (**Warning**: with a filename, these methods will overwrite existing files silently.)

Individual pages, meta-data, other output formats, ...

If you want more than a single PDF, the *render()* method gives you a *Document* object with access to individual *Page* objects. Thus you can get the number of pages, their size[1], the details of hyperlinks and bookmarks, etc. Documents also have *write_pdf()* and *write_png()* methods, and you can get a subset of the pages with *copy()*. Finally, for ultimate control, *paint()* individual pages anywhere on any type of cairo surface.

See the *Python API* for details. A few random example:

```
# Write odd and even pages separately:
#   Lists count from 0 but page numbers usually from 1
#   [::2] is a slice of even list indexes but odd-numbered pages.
document.copy(document.pages[::2]).write_pdf('odd_pages.pdf')
document.copy(document.pages[1::2]).write_pdf('even_pages.pdf')
```

```
# Write one PNG image per page:
for i, page in enumerate(document.pages):
    document.copy([page]).write_png('page_%s.png' % i)
```

```
# Some previous versions of WeasyPrint had a method like this:
def get_png_pages(document):
    """Yield (png_bytes, width, height) tuples."""
    for page in document.pages:
        yield document.copy([page]).write_png()
```

```
# Print the outline of the document.
# Output on http://www.w3.org/TR/CSS21/intro.html
#     1. Introduction to CSS 2.1 (page 2)
#       1. A brief CSS 2.1 tutorial for HTML (page 2)
#       2. A brief CSS 2.1 tutorial for XML (page 5)
#       3. The CSS 2.1 processing model (page 6)
#         1. The canvas (page 7)
#         2. CSS 2.1 addressing model (page 7)
#       4. CSS design principles (page 8)
def print_outline(bookmarks, indent=0):
    for i, (label, (page, _, _), children) in enumerate(bookmarks, 1):
        print('%s%d. %s (page %d)' % (
            ' ' * indent, i, label.lstrip('0123456789. '), page))
        print_outline(children, indent + 2)
print_outline(document.make_bookmark_tree())
```

```
# PostScript on standard output:
surface = cairo.PSSurface(sys.stdout, 1, 1)
context = cairo.Context(surface)
for page in document.pages:
    # 0.75 = 72 PostScript point per inch / 96 CSS pixel per inch
```

(continues on next page)

[1] Pages in the same document do not always have the same size.

(continued from previous page)

```
    surface.set_size(page.width * 0.75, page.height * 0.75)
    page.paint(context, scale=0.75)
    surface.show_page()
surface.finish()
```

URL fetchers

WeasyPrint goes through a *URL fetcher* to fetch external resources such as images or CSS stylesheets. The default fetcher can natively open file and HTTP URLs, but the HTTP client does not support advanced features like cookies or authentication. This can be worked-around by passing a custom `url_fetcher` callable to the `HTML` or `CSS` classes. It must have the same signature as *`default_url_fetcher()`*.

Custom fetchers can choose to handle some URLs and defer others to the default fetcher:

```
from weasyprint import default_url_fetcher, HTML

def my_fetcher(url):
    if url.startswith('graph:'):
        graph_data = map(float, url[6:].split(','))
        return dict(string=generate_graph(graph_data),
                    mime_type='image/png')
    else:
        return weasyprint.default_url_fetcher(url)

source = '<img src="graph:42,10.3,87">'
HTML(string=source, url_fetcher=my_fetcher).write_pdf('out.pdf')
```

Flask-WeasyPrint makes use of a custom URL fetcher to integrate WeasyPrint with a Flask application and short-cut the network for resources that are within the same application.

Logging

Most errors (unsupported CSS property, missing image, ...) are not fatal and will not prevent a document from being rendered.

WeasyPrint uses the `logging` module from the Python standard library to log these errors and let you know about them. When WeasyPrint is launched in a terminal, logged messaged will go to *stderr* by default. You can change that by configuring the `weasyprint` logger object:

```
import logging
logger = logging.getLogger('weasyprint')
logger.addHandler(logging.FileHandler('/path/to/weasyprint.log'))
```

The `INFO` level is used to report the rendering progress. It is useful to get feedback when WeasyPrint is launched in a terminal (using the `--verbose` option), or to give this feedback to end users when used as a library. To catch these logs, you can for example use a filter:

```
import logging

class LoggerFilter(logging.Filter):
    def filter(self, record):
        if record.level == logging.INFO:
            print(record.getMessage())
            return False
```

(continues on next page)

(continued from previous page)

```
logger = logging.getLogger('weasyprint')
logger.addFilter(LoggerFilter())
```

See the documentation of the `logging` module for details.

1.2.3 WeasyPrint Tools

WeasyPrint provides two very limited tools, helping users to play with WeasyPrint, test it, and understand how to use it as a library.

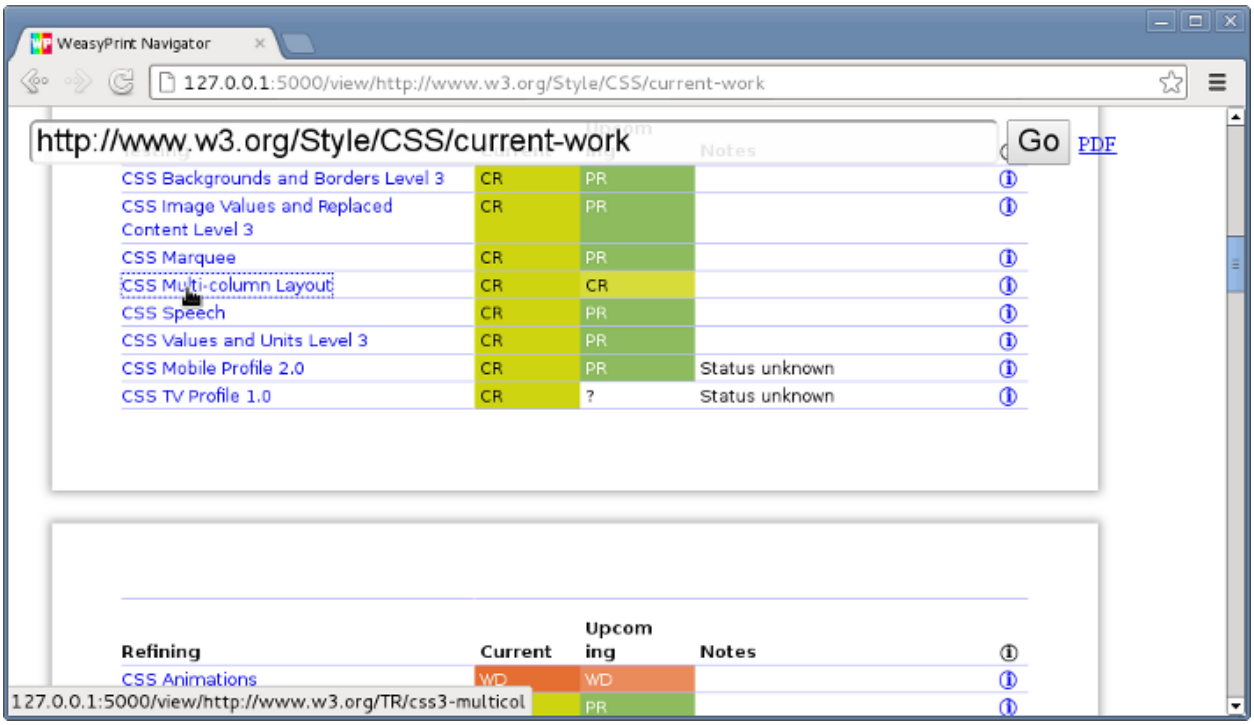
These tools are just "toys" and are not intended to be significantly improved in the future.

WeasyPrint Navigator

WeasyPrint Navigator is a web browser running in your web browser. Start it with:

```
python -m weasyprint.tools.navigator
```

. . . and open your browser at http://127.0.0.1:5000/.



It does not support cookies, forms, or many other things that you would expect from a "real" browser. It only shows the PNG output from WeasyPrint with overlaid clickable hyperlinks. It is mostly useful for playing and testing.

WeasyPrint Renderer

WeasyPrint Renderer is a web app providing on the same web page a textarea where you can type an HTML/CSS document, and this document rendered by WeasyPrint as a PNG image. Start it with:

```
python -m weasyprint.tools.renderer
```

... and open your browser at http://127.0.0.1:5000/.

1.2.4 Security

When used with untrusted HTMl or untrusted CSS, WeasyPrint can meet security problems. You will need extra configuration in your Python application to avoid high memory use, endless renderings or local files leaks.

This section has been added thanks to the very useful reports and advice from Raz Becker.

Long renderings

WeasyPrint is pretty slow and can take a long time to render long documents or specially crafted HTML pages.

When WeasyPrint used on a server with HTML or CSS files from untrusted sources, this problem can lead to very long time renderings, with processes with high CPU and memory use. Even small documents may lead to really long rendering times, restricting HTML document size is not enough.

If you use WeasyPrint on a server with HTML or CSS samples coming from untrusted users, you should:

- limit rendering time and memory use of your process, for example using `evil-reload-on-as` and `harakiri` options if you use uWSGI,
- limit memory use at the OS level, for example with `ulimit` on Linux,
- automatically kill the process when it uses too much memory or when the rendering time is too high, by regularly launching a script to do so if no better option is available,
- truncate and sanitize HTML and CSS input to avoid very long documents and access to external URLs.

Infinite requests

WeasyPrint can reach files on the network, for example using `http://` URIs. For various reasons, HTTP requests may take a long time and lead to problems similar to *Long renderings*.

WeasyPrint has a default timeout of 10 seconds for HTTP, HTTPS and FTP resources. This timeout has no effect with other protocols, including access to `file://` URIs.

If you use WeasyPrint on a server with HTML or CSS samples coming from untrusted users, or need to reach network resources, you should:

- use a custom URL fetcher,
- follow solutions listed in *Long renderings*.

Infinite loops

WeasyPrint has been hit by a large number of bugs, including infinite loops. Specially crafted HTML and CSS files can quite easily lead to infinite loops and infinite rendering times.

If you use WeasyPrint on a server with HTML or CSS samples coming from untrusted users, you should:

- follow solutions listed in *Long renderings*.

Huge values

WeasyPrint doesn't restrict integer and float values used in CSS. Using huge values for some properties (page sizes, font sizes, block sizes) can lead to various problems, including infinite rendering times, huge PDF files, high memory use and crashes.

This problem is really hard to avoid. Even parsing CSS stylesheets and searching for huge values is not enough, as it is quite easy to trick CSS pre-processors using relative units (`em` and `%` for example).

If you use WeasyPrint on a server with HTML or CSS samples coming from untrusted users, you should:

- follow solutions listed in *Long renderings*.

Access to local files

As any web renderer, WeasyPrint can reach files on the local filesystem using `file://` URIs. These files can be shown in `img` or `embed` tags for example.

When WeasyPrint used on a server with HTML or CSS files from untrusted sources, this feature may be used to know if files are present on the server filesystem, and to embed them in generated documents.

Unix-like systems also have special local files with infinite size, like `/dev/urandom`. Referencing these files in HTML or CSS files obviously lead to infinite time renderings.

If you use WeasyPrint on a server with HTML or CSS samples coming from untrusted users, you should:

- restrict your process access to trusted files using sandboxing solutions,
- use a custom URL fetcher that doesn't allow `file://` URLs or filters access depending on given paths.
- follow solutions listed in *Long renderings*.

System information leaks

WeasyPrint relies on many libraries that can leak hardware and software information. Even when this information looks useless, it can be used by attackers to exploit other security breaches.

Leaks can include (but are not restricted to):

- locally installed fonts (using `font-family` and `@font-face`),
- network configuration (IPv4 and IPv6 support, IP addressing, firewall configuration, using `http://` URIs and tracking time used to render documents),
- hardware and software used for graphical rendering (as Cairo renderings can change with CPU and GPU features),
- Python, Cairo, Pango and other libraries versions (implementation details lead to different renderings).

SVG images

WeasyPrint relies on CairoSVG to render SVG files. CairoSVG more or less suffers from the same problems as the ones listed here for WeasyPrint.

Security advices apply for untrusted SVG files as they apply for untrusted HTML and CSS documents.

Note that WeasyPrint gives CairoSVG its URL fetcher.

1.2.5 Errors

If you get an exception during rendering, it is probably a bug in WeasyPrint. Please copy the full traceback and report it on our issue tracker.

1.2.6 Stylesheet origins

HTML documents are rendered with stylesheets from three *origins*:

- The HTML5 user agent stylesheet (defines the default appearance of HTML elements);
- Author stylesheets embedded in the document in `<style>` elements or linked by `<link rel=stylesheet>` elements;
- User stylesheets provided in the API.

Keep in mind that *user* stylesheets have a lower priority than *author* stylesheets in the cascade, unless you use !important in declarations to raise their priority.

1.3 API

1.3.1 API stability

Everything described here is considered "public": this is what you can rely on. We will try to maintain backward-compatibility, although there is no hard promise until version 1.0.

Anything else should not be used outside of WeasyPrint itself: we reserve the right to change it or remove it at any point. Use it at your own risk, or have dependency to a specific WeasyPrint version in your `setup.py` or `requirements.txt` file.

1.3.2 Command-line API

`weasyprint.__main__.`**`main`**(*argv=sys.argv*)

The `weasyprint` program takes at least two arguments:

```
weasyprint [options] <input> <output>
```

The input is a filename or URL to an HTML document, or − to read HTML from stdin. The output is a filename, or − to write to stdout.

Options can be mixed anywhere before, between, or after the input and output:

`-e` `<input_encoding>`**`, --encoding`** `<input_encoding>`
Force the input character encoding (e.g. `-e utf8`).

`-f` `<output_format>`**`, --format`** `<output_format>`
Choose the output file format among PDF and PNG (e.g. `-f png`). Required if the output is not a `.pdf` or `.png` filename.

`-s` `<filename_or_URL>`**`, --stylesheet`** `<filename_or_URL>`
Filename or URL of a user CSS stylesheet (see *Stylesheet origins*) to add to the document (e.g. `-s print.css`). Multiple stylesheets are allowed.

`-r` `<dpi>`**`, --resolution`** `<dpi>`
For PNG output only. Set the resolution in PNG pixel per CSS inch. Defaults to 96, which means that PNG pixels match CSS pixels.

--base-url <URL>
Set the base for relative URLs in the HTML input. Defaults to the input's own URL, or the current directory for stdin.

-m <type>, **--media-type** <type>
Set the media type to use for `@media`. Defaults to `print`.

-a <file>, **--attachment** <file>
Adds an attachment to the document. The attachment is included in the PDF output. This option can be used multiple times.

-p, --presentational-hints
Follow HTML presentational hints.

--version
Show the version number. Other options and arguments are ignored.

-h, --help
Show the command-line usage. Other options and arguments are ignored.

1.3.3 Python API

class weasyprint.**HTML**(*input*, ***kwargs*)
Represents an HTML document parsed by html5lib.

You can just create an instance with a positional argument: `doc = HTML(something)` The class will try to guess if the input is a filename, an absolute URL, or a file-like object.

Alternatively, use **one** named argument so that no guessing is involved:

Parameters

- **filename** – A filename, relative to the current directory, or absolute.
- **url** – An absolute, fully qualified URL.
- **file_obj** – A file-like: any object with a `read()` method.
- **string** – A string of HTML source. (This argument must be named.)

Specifying multiple inputs is an error: `HTML(filename="foo.html", url="localhost://bar.html")` will raise a TypeError.

You can also pass optional named arguments:

Parameters

- **encoding** – Force the source character encoding.
- **base_url** – The base used to resolve relative URLs (e.g. in `<img src="../foo.png">`). If not provided, try to use the input filename, URL, or `name` attribute of file-like objects.
- **url_fetcher** – A function or other callable with the same signature as *default_url_fetcher()* called to fetch external resources such as stylesheets and images. (See *URL fetchers*.)
- **media_type** – The media type to use for `@media`. Defaults to `'print'`. **Note:** In some cases like `HTML(string=foo)` relative URLs will be invalid if `base_url` is not provided.

render(*stylesheets=None*, *enable_hinting=False*, *presentational_hints=False*, *font_config=None*)
Lay out and paginate the document, but do not (yet) export it to PDF or another format.

This returns a *Document* object which provides access to individual pages and various meta-data. See *write_pdf()* to get a PDF directly.

New in version 0.15.

Parameters

- **stylesheets** – An optional list of user stylesheets. List elements are *CSS* objects, filenames, URLs, or file-like objects. (See *Stylesheet origins*.)
- **enable_hinting** (*bool*) – Whether text, borders and background should be *hinted* to fall at device pixel boundaries. Should be enabled for pixel-based output (like PNG) but not for vector-based output (like PDF).
- **presentational_hints** (*bool*) – Whether HTML presentational hints are followed.
- **font_config** (FontConfiguration) – A font configuration handling @font-face rules.

Returns A *Document* object.

write_pdf(*target=None*, *stylesheets=None*, *zoom=1*, *attachments=None*, *presentational_hints=False*, *font_config=None*)
Render the document to a PDF file.

This is a shortcut for calling *render()*, then *Document.write_pdf()*.

Parameters

- **target** – A filename, file-like object, or None.
- **stylesheets** – An optional list of user stylesheets. The list's elements are *CSS* objects, filenames, URLs, or file-like objects. (See *Stylesheet origins*.)
- **zoom** (*float*) – The zoom factor in PDF units per CSS units. **Warning**: All CSS units are affected, including physical units like cm and named sizes like A4. For values other than 1, the physical CSS units will thus be "wrong".
- **attachments** – A list of additional file attachments for the generated PDF document or None. The list's elements are Attachment objects, filenames, URLs or file-like objects.
- **presentational_hints** (*bool*) – Whether HTML presentational hints are followed.
- **font_config** (FontConfiguration) – A font configuration handling @font-face rules.

Returns The PDF as byte string if target is not provided or None, otherwise None (the PDF is written to target).

write_png(*target=None*, *stylesheets=None*, *resolution=96*, *presentational_hints=False*, *font_config=None*)
Paint the pages vertically to a single PNG image.

There is no decoration around pages other than those specified in CSS with @page rules. The final image is as wide as the widest page. Each page is below the previous one, centered horizontally.

This is a shortcut for calling *render()*, then *Document.write_png()*.

Parameters

- **target** – A filename, file-like object, or None.

- **stylesheets** – An optional list of user stylesheets. The list's elements are *CSS* objects, filenames, URLs, or file-like objects. (See *Stylesheet origins*.)
- **resolution** (*float*) – The output resolution in PNG pixels per CSS inch. At 96 dpi (the default), PNG pixels match the CSS `px` unit.
- **presentational_hints** (*bool*) – Whether HTML presentational hints are followed.
- **font_config** (`FontConfiguration`) – A font configuration handling @font-face rules.

Returns The image as byte string if `target` is not provided or `None`, otherwise `None` (the image is written to `target`.)

class `weasyprint.`**CSS**(*input*, ***kwargs*)

Represents a CSS stylesheet parsed by tinycss2.

An instance is created in the same way as *HTML*, except that the `tree` argument is not available. All other arguments are the same.

An additional argument called `font_config` must be provided to handle `@font-config` rules. The same `fonts.FontConfiguration` object must be used for different `CSS` objects applied to the same document.

`CSS` objects have no public attribute or method. They are only meant to be used in the *write_pdf()*, *write_png()* and *render()* methods of *HTML* objects.

`weasyprint.`**default_url_fetcher**(*url*, *timeout=10*)

Fetch an external resource such as an image or stylesheet.

Another callable with the same signature can be given as the `url_fetcher` argument to *HTML* or *CSS*. (See *URL fetchers*.)

Parameters **url** (*Unicode string*) – The URL of the resource to fetch.

Raises An exception indicating failure, e.g. `ValueError` on syntactically invalid URL.

Returns

A dict with the following keys:

- One of `string` (a byte string) or `file_obj` (a file-like object)
- Optionally: `mime_type`, a MIME type extracted e.g. from a *Content-Type* header. If not provided, the type is guessed from the file extension in the URL.
- Optionally: `encoding`, a character encoding extracted e.g. from a *charset* parameter in a *Content-Type* header
- Optionally: `redirected_url`, the actual URL of the resource if there were e.g. HTTP redirects.
- Optionally: `filename`, the filename of the resource. Usually derived from the *filename* parameter in a *Content-Disposition* header

If a `file_obj` key is given, it is the caller's responsibility to call `file_obj.close()`.

class `weasyprint.document.`**Document**(*pages*, *metadata*, *url_fetcher*, *font_config*)

A rendered document, with access to individual pages ready to be painted on any cairo surfaces.

Typically obtained from *HTML.render()*, but can also be instantiated directly with a list of *pages*, a set of *metadata*, and a `url_fetcher`.

pages = None

A list of *Page* objects.

metadata = None
A *DocumentMetadata* object. Contains information that does not belong to a specific page but to the whole document.

url_fetcher = None
A `url_fetcher` for resources that have to be read when writing the output.

copy(*pages='all'*)
Take a subset of the pages.

Parameters **pages** – An iterable of *Page* objects from *pages*.

Returns A new *Document* object.

Examples:

Write two PDF files for odd-numbered and even-numbered pages:

```
# Python lists count from 0 but pages are numbered from 1.
# [::2] is a slice of even list indexes but odd-numbered pages.
document.copy(document.pages[::2]).write_pdf('odd_pages.pdf')
document.copy(document.pages[1::2]).write_pdf('even_pages.pdf')
```

Write each page to a numbred PNG file:

```
for i, page in enumerate(document.pages):
    document.copy(page).write_png('page_%s.png' % i)
```

Combine multiple documents into one PDF file, using metadata from the first:

```
all_pages = [p for p in doc.pages for doc in documents]
documents[0].copy(all_pages).write_pdf('combined.pdf')
```

resolve_links()
Resolve internal hyperlinks.

Links to a missing anchor are removed with a warning.

If multiple anchors have the same name, the first one is used.

Returns A generator yielding lists (one per page) like `Page.links`, except that `target` for internal hyperlinks is `(page_number, x, y)` instead of an anchor name. The page number is a 0-based index into the *pages* list, and `x, y` are in CSS pixels from the top-left of the page.

make_bookmark_tree()
Make a tree of all bookmarks in the document.

Returns a list of bookmark subtrees. A subtree is `(label, target, children)`. `label` is a string, `target` is `(page_number, x, y)` like in *resolve_links()*, and `children` is a list of child subtrees.

add_hyperlinks(*links*, *anchors*, *context*, *scale*)
Include hyperlinks in current page.

write_pdf(*target=None*, *zoom=1*, *attachments=None*)
Paint the pages in a PDF file, with meta-data.

PDF files written directly by cairo do not have meta-data such as bookmarks/outlines and hyperlinks.

Parameters

- **target** – A filename, file-like object, or `None`.

- **zoom** (*float*) – The zoom factor in PDF units per CSS units. **Warning**: All CSS units are affected, including physical units like `cm` and named sizes like `A4`. For values other than 1, the physical CSS units will thus be "wrong".
- **attachments** – A list of additional file attachments for the generated PDF document or `None`. The list's elements are `Attachment` objects, filenames, URLs, or file-like objects.

Returns The PDF as byte string if `target` is `None`, otherwise `None` (the PDF is written to `target`).

write_png(*target=None*, *resolution=96*)

Paint the pages vertically to a single PNG image.

There is no decoration around pages other than those specified in CSS with `@page` rules. The final image is as wide as the widest page. Each page is below the previous one, centered horizontally.

Parameters

- **target** – A filename, file-like object, or `None`.
- **resolution** (*float*) – The output resolution in PNG pixels per CSS inch. At 96 dpi (the default), PNG pixels match the CSS `px` unit.

Returns A `(png_bytes, png_width, png_height)` tuple. `png_bytes` is a byte string if `target` is `None`, otherwise `None` (the image is written to `target`). `png_width` and `png_height` are the size of the final image, in PNG pixels.

class weasyprint.document.**DocumentMetadata**(*title=None*, *authors=None*, *description=None*, *keywords=None*, *generator=None*, *created=None*, *modified=None*, *attachments=None*)

Contains meta-information about a *Document* that belongs to the whole document rather than specific pages.

New attributes may be added in future versions of WeasyPrint.

title = None

The title of the document, as a string or `None`. Extracted from the `<title>` element in HTML and written to the `/Title` info field in PDF.

authors = None

The authors of the document as a list of strings. Extracted from the `<meta name=author>` elements in HTML and written to the `/Author` info field in PDF.

description = None

The description of the document, as a string or `None`. Extracted from the `<meta name=description>` element in HTML and written to the `/Subject` info field in PDF.

keywords = None

Keywords associated with the document, as a list of strings. (Defaults to the empty list.) Extracted from `<meta name=keywords>` elements in HTML and written to the `/Keywords` info field in PDF.

generator = None

The name of one of the software packages used to generate the document, as a string or `None`. Extracted from the `<meta name=generator>` element in HTML and written to the `/Creator` info field in PDF.

created = None

The creation date of the document, as a string or `None`. Dates are in one of the six formats specified in W3C's profile of ISO 8601. Extracted from the `<meta name=dcterms.created>` element in HTML and written to the `/CreationDate` info field in PDF.

modified = None
The modification date of the document, as a string or None. Dates are in one of the six formats specified in W3C's profile of ISO 8601. Extracted from the <meta name=dcterms.modified> element in HTML and written to the /ModDate info field in PDF.

attachments = None
File attachments as a list of tuples of URL and a description or None. Extracted from the <link rel=attachment> elements in HTML and written to the /EmbeddedFiles dictionary in PDF.

class weasyprint.document.**Page**
Represents a single rendered page.

New in version 0.15.

Should be obtained from *Document.pages* but not instantiated directly.

width = None
The page width, including margins, in CSS pixels.

height = None
The page height, including margins, in CSS pixels.

bleed = None
The page bleed width, in CSS pixels.

paint(*cairo_context*, *left_x=0*, *top_y=0*, *scale=1*, *clip=False*)
Paint the page in cairo, on any type of surface.

Parameters

- **cairo_context** – Any cairocffi.Context object.
- **left_x** (*float*) – X coordinate of the left of the page, in cairo user units.
- **top_y** (*float*) – Y coordinate of the top of the page, in cairo user units.
- **scale** (*float*) – Zoom scale in cairo user units per CSS pixel.
- **clip** (*bool*) – Whether to clip/cut content outside the page. If false or not provided, content can overflow.

1.4 Features

This page is for WeasyPrint 43. See *changelog* for older versions.

1.4.1 URLs

WeasyPrint can read normal files, HTTP, FTP and data URIs. It will follow HTTP redirects but more advanced features like cookies and authentication are currently not supported, although a custom *url fetcher* can help.

1.4.2 HTML

Many HTML elements are implemented in CSS through the HTML5 User-Agent stylesheet.

Some elements need special treatment:

- The <base> element, if present, determines the base for relative URLs.
- CSS stylesheets can be embedded in <style> elements or linked by <link rel=stylesheet> elements.

- `<img>`, `<embed>` or `<object>` elements accept images either in raster formats supported by GdkPixbuf (including PNG, JPEG, GIF, ...) or in SVG with CairoSVG. SVG images are not rasterized but rendered as vectors in the PDF output.

HTML presentational hints are not supported by default, but most of them can be supported:

- by using the `--presentational-hints` CLI parameter, or
- by setting the `presentational_hints` parameter of the `HTML.render` or `HTML.write_*` methods to `True`.

Presentational hints include a wide array of attributes that direct styling in HTML, including font `color` and `size`, list attributes like `type` and `start`, various table alignment attributes, and others. If the document generated by WeasyPrint is missing some of the features you expect from the HTML, try to enable this option.

1.4.3 PDF

In addition to text, raster and vector graphics, WeasyPrint's PDF files can contain hyperlinks, bookmarks and attachments.

Hyperlinks will be clickable in PDF viewers that support them. They can be either internal, to another part of the same document (eg. `<a href="#pdf">`) or external, to an URL. External links are resolved to absolute URLs: `<a href="/news/">` on the WeasyPrint website would always point to http://weasyprint.org/news/ in PDF files.

PDF bookmarks are also called outlines and are generally shown in a sidebar. Clicking on an entry scrolls the matching part of the document into view. By default all `<h1>` to `<h6>` titles generate bookmarks, but this can be controlled with CSS (see *Bookmarks*.)

Attachments are related files, embedded in the PDF itself. They can be specified through `<link rel=attachment>` elements to add resources globally or through regular links with `<a rel=attachment>` to attach a resource that can be saved by clicking on said link. The `title` attribute can be used as description of the attachment.

1.4.4 Fonts

WeasyPrint can use any font that Pango can find installed on the system. Fonts are automatically embedded in PDF files.

On Linux, Pango uses fontconfig to access fonts. You can list the available fonts thanks to the `fc-list` command, and know which font is matched by a given pattern thanks to `fc-match`. Copying a font file into the `~/.local/share/fonts` or `~/.fonts` directory is generally enough to install a new font. WeasyPrint should support any font format handled by FreeType (any format widely used except WOFF2).

On Windows and macOS, **Pango >= 1.38** is required to use fontconfig and FreeType like it does on Linux. Both, `fc-list` and `fc-match` probably will be present, too. Installing new fonts on your system as usual should make them available to Pango.

Otherwise (Pango < 1.38) on Windows and macOS, the native font-managing libraries are used. You must then use the tools provided by your OS to know which fonts are available. WeasyPrint should support any font format that's supported by the operating system.

1.4.5 CSS

WeasyPrint supports many of the CSS specifications written by the W3C. You will find in this chapter a comprehensive list of the specifications or drafts with at least one feature implemented in WeasyPrint.

The results of some of the test suites provided by the W3C are also available at test.weasyprint.org. This website uses a tool called WeasySuite that can be useful if you want to implement new features in WeasyPrint.

CSS Level 2 Revision 1

The CSS Level 2 Revision 1 specification, best known as CSS 2.1, is pretty well supported by WeasyPrint. Since version 0.11, it passes the famous Acid2 Test.

The CSS 2.1 features listed here are **not** supported:

- The ::first-line pseudo-element.
- On tables: visibility: collapse.
- Minimum and maximum height on table-related boxes.
- Minimum and maximum width and height on page-margin boxes.
- Conforming font matching algorithm. Currently `font-family` is passed as-is to Pango.
- Right-to-left or bi-directional text.
- System colors and system fonts. The former are deprecated in CSS Color Module Level 3.

To the best of our knowledge, everything else that applies to the print media **is** supported. Please report a bug if you find this list incomplete.

Selectors Level 3

With the exceptions noted here, all Selectors Level 3 are supported.

PDF is generally not interactive. The `:hover`, `:active`, `:focus`, `:target` and `:visited` pseudo-classes are accepted as valid but never match anything.

CSS Text Module Level 3 / 4

The CSS Text Module Level 3 and CSS Text Module Level 4 are working drafts defining "properties for text manipulation" and covering "line breaking, justification and alignment, white space handling, and text transformation".

Among their features, some are already included in CSS 2.1, sometimes with missing or different values (`text-indent`, `text-align`, `letter-spacing`, `word-spacing`, `text-transform`, `white-space`).

New properties defined in Level 3 are supported:

- the `overflow-wrap` property replacing `word-wrap`;
- the `full-width` value of the `text-transform` property; and
- the `tab-space` property.

Experimental properties controling hyphenation are supported by WeasyPrint:

- `hyphens`,
- `hyphenate-character`,
- `hyphenate-limit-chars`, and
- `hyphenate-limit-zone`.

To get automatic hyphenation, you to set it to `auto` *and* have the `lang` HTML attribute set to one of the languages supported by Pyphen.

```
<!doctype html>
<html lang=en>
<style>
  html { hyphens: auto }
</style>
...
```

Automatic hyphenation can be disabled again with the `manual` value:

```
html { hyphens: auto }
a[href]::after { content: ' [' attr(href) ']'; hyphens: manual }
```

The other features provided by CSS Text Module Level 3 are **not** supported:

- the `line-break` and `word-break` properties;
- the `start`, `end`, `match-parent` and `start end` values of the `text-align` property;
- the `text-align-last` and `text-justify` properties; and
- the `text-indent` and `hanging-punctuation` properties.

The other features provided by CSS Text Module Level 4 are **not** supported:

- the `text-space-collapse` and `text-space-trim` properties;
- the `text-wrap`, `wrap-before`, `wrap-after` and `wrap-inside` properties;
- the `pre-wrap-auto` value of the `white-space` property; and
- the `text-spacing` property.

CSS Fonts Module Level 3

The *CSS Fonts Module Level 3* is a candidate recommendation describing "how font properties are specified and how font resources are loaded dynamically".

WeasyPrint supports the `font-size`, `font-stretch`, `font-style` and `font-weight` properties, coming from CSS 2.1.

WeasyPrint also supports the following font features added in Level 3: - `font-kerning`, - `font-variant-ligatures`, - `font-variant-position`, - `font-variant-caps`, - `font-variant-numeric`, - `font-variant-east-asian`, - `font-feature-settings`, and - `font-language-override`.

`font-family` is supported. The string is given to Pango that tries to find a matching font in a way different from what is defined in the recommendation, but that should not be a problem for common use.

The shorthand `font` and `font-variant` properties are supported.

WeasyPrint supports the `@font-face` rule, provided that Pango >= 1.38 is installed.

WeasyPrint does **not** support the `@font-feature-values` rule and the values of `font-variant-alternates` other than `normal` and `historical-forms`.

The `font-variant-caps` property is supported but needs the small-caps variant of the font to be installed. WeasyPrint does **not** simulate missing small-caps fonts.

CSS Paged Media Module Level 3

The CSS Paged Media Module Level 3 is a working draft including features for paged media "describing how:

- page breaks are created and avoided;
- the page properties such as size, orientation, margins, border, and padding are specified;
- headers and footers are established within the page margins;
- content such as page counters are placed in the headers and footers; and
- orphans and widows can be controlled."

All the features of this draft are available, including:

- the `@page` rule and the `:left`, `:right`, `:first` and `:blank` selectors;
- the page margin boxes;
- the page-based counters (with known limitations #93);
- the page `size`, `bleed` and `marks` properties;
- the named pages.

CSS Generated Content for Paged Media Module

The CSS Generated Content for Paged Media Module (GCPM) is a working draft defining "new properties and values, so that authors may bring new techniques (running headers and footers, footnotes, leaders, bookmarks) to paged media".

Three features from this module have been implemented in WeasyPrint.

The first feature is PDF bookmarks. Using the experimental `bookmark-level` and `bookmark-level` properties, you can add bookmarks that will be available in your PDF reader.

Bookmarks have already been added in the WeasyPrint's user agent stylesheet, so your generated documents will automatically have bookmarks on headers (from `<h1>` to `<h6>`). But for example, if you have only one top-level `<h1>` and do not wish to include it in the bookmarks, add this in your stylesheet:

```
h1 { bookmark-level: none }
```

The second feature is Named strings. You can define strings related to the first or last element of a type present on a page, and display these strings in page borders. This feature is really useful to add the title of the current chapter at the top of the pages of a book for example.

The named strings can embed static strings, counters, cross-references, tag contents and tag attributes.

```
@top-center { content: string(chapter); }
h2 { string-set: chapter "Current chapter: " content() }
```

The third feature is internal Cross-references, which makes it possible to retrieve counter or content values from targets (anchors or ids) in the current document:

```
a::after {
  content: ", on page " target-counter(attr(href), page);
}
a::after {
  content: ", see " target-text(attr(href));
}
```

In particular, `target-counter()` and `target-text()` are useful when it comes to tables of contents, see an example.

The other features of GCPM are **not** implemented:

- running elements (`running()` and `element()`);
- footnotes (`float: footnote`, `footnote-display`, `footnote` counter, `::footnote-call`, `::footnote-marker`, `@footnote` rule, `footnote-policy`);
- page selectors and page groups (`:nth()` pseudo-class);
- leaders (`content: leader()`);
- bookmark states (`bookmark-state`).

CSS Color Module Level 3

The CSS Color Module Level 3 is a recommendation defining "CSS properties which allow authors to specify the foreground color and opacity of an element". Its main goal is to specify how colors are defined, including color keywords and the `#rgb`, `#rrggbb`, `rgb()`, `rgba()`, `hsl()`, `hsla()` syntaxes. Opacity and alpha compositing are also defined in this document.

This recommendation is fully implemented in WeasyPrint, except the deprecated System Colors.

CSS Transforms Module Level 1

The CSS Transforms Module Level 1 working draft "describes a coordinate system within each element is positioned. This coordinate space can be modified with the transform property. Using transform, elements can be translated, rotated and scaled in two or three dimensional space."

WeasyPrint supports the `transform` and `transform-origin` properties, and all the 2D transformations (`matrix`, `rotate`, `translate(X|Y)?`, `scale(X|Y)?`, `skew(X|Y)?`).

WeasyPrint does **not** support the `transform-style`, `perspective`, `perspective-origin` and `backface-visibility` properties, and all the 3D transformations (`matrix3d`, `rotate(3d|X|Y|Z)`, `translate(3d|Z)`, `scale(3d|Z)`).

CSS Backgrounds and Borders Module Level 3

The *CSS Backgrounds and Borders Module Level 3* is a candidate recommendation defining properties dealing "with the decoration of the border area and with the background of the content, padding and border areas".

The border part of this module is supported, as it is already included in the the CSS 2.1 specification.

WeasyPrint supports the background part of this module (allowing multiple background layers per box), including the `background`, `background-color`, `background-image`, `background-repeat`, `background-attachment`, `background-position`, `background-clip`, `background-origin` and `background-size` properties.

WeasyPrint also supports the rounded corners part of this module, including the `border-radius` property.

WeasyPrint does **not** support the border images part of this module, including the `border-image`, `border-image-source`, `border-image-slice`, `border-image-width`, `border-image-outset` and `border-image-repeat` properties.

WeasyPrint does **not** support the box shadow part of this module, including the `box-shadow` property. This feature has been implemented in a git branch that is not released, as it relies on raster implementation of shadows.

CSS Image Values and Replaced Content Module Level 3 / 4

The Image Values and Replaced Content Module Level 3 is a candidate recommendation introducing "additional ways of representing 2D images, for example as a list of URIs denoting fallbacks, or as a gradient", defining "several properties for manipulating raster images and for sizing or positioning replaced elements" and "generic sizing algorithm for replaced elements".

The Image Values and Replaced Content Module Level 4 is a working draft on the same subject.

The `linear-gradient()`, `radial-gradient()` and `repeating-radial-gradient()` properties are supported as background images.

The the `url()` notation is supported, but the `image()` notation is **not** supported for background images.

The `from-image` and `snap` values of the `image-resolution` property are **not** supported, but the `resolution` value is supported.

The `image-rendering` property is supported.

The `image-orientation`, `object-fit` and `object-position` are **not** supported.

CSS Basic User Interface Module Level 3

The CSS Basic User Interface Module Level 3 also known as CSS3 UI is a candidate recommendation describing "CSS properties which enable authors to style user interface related properties and values."

Only one new property defined in this document is implemented in WeasyPrint: the `box-sizing` property.

Some of the properties do not apply for WeasyPrint: `cursor`, `resize`, `caret-color`, `nav-(up|right|down|left)`.

The other properties are **not** implemented: `outline-offset` and `text-overflow`.

CSS Values and Units Module Level 3

The CSS Values and Units Module Level 3 defines various units and keywords used in "value definition field of each CSS property".

The `initial` and `inherit` CSS-wide keywords are supported, but the `unset` keyword is **not** supported.

Quoted strings, URLs and numeric data types are supported.

Font-related lengths (`em`, `ex`, `ch`, `rem`), absolute lengths (`cm`, `mm`, `q`, `in`, `pt`, `pc`, `px`), angles (`rad`, `grad`, `turn`, `deg`), resolutions (`dpi`, `dpcm`, `dppx`) are supported.

The `attr()` functional notation is allowed in the `content` and `string-set` properties.

Viewport-percentage lengths (`vw`, `vh`, `vmin`, `vmax`) are **not** supported.

CSS Multi-column Layout Module

The CSS Multi-column Layout Module "describes multi-column layouts in CSS, a style sheet language for the web. Using functionality described in the specification, content can be flowed into multiple columns with a gap and a rule between them."

Simple multi-column layouts are supported in WeasyPrint. Features such as constrained height, spanning columns or column breaks are **not** supported. Pagination and overflow are not seriously tested.

The `column-width` and `column-count` properties, and the `columns` shorthand property are supported.

The `column-gap`, `column-rule-color`, `column-rule-style` and `column-rule-width` properties, and the `column-rule` shorthand property are supported.

The `break-before`, `break-after` and `break-inside` properties are **not** supported.

The `column-span` property is **not** supported.

The `column-fill` property is supported, with a column balancing algorithm that should be efficient with simple cases.

1.5 Hacking WeasyPrint

Assuming you already have the *dependencies*, install the development version of WeasyPrint:

```
git clone git://github.com/Kozea/WeasyPrint.git
cd WeasyPrint
python3 -m venv env
. env/bin/activate
pip install -e .[doc,test]
weasyprint --help
```

This will install WeasyPrint in "editable" mode (which means that you don't need to re-install it every time you make a change in the source code) as well as pytest and Sphinx.

Lastly, in order to pass unit tests, your system must have as default font any font with a condensed variant (i.e. DejaVu) - typically installable via your distro's packaging system.

1.5.1 Documentation changes

The documentation lives in the `docs` directory, but API section references docstrings in the source code. Run `python setup.py build_sphinx` to rebuild the documentation and get the output in `docs/_build/html`. The website version is updated automatically when we push to master on GitHub.

1.5.2 Code changes

Use the `python setup.py test` command from the `WeasyPrint` directory to run the test suite.

Please report any bugs/feature requests and submit patches/pull requests on Github.

1.5.3 Dive into the source

The rest of this document is a high-level overview of WeasyPrint's source code. For more details, see the various docstrings or even the code itself. When in doubt, feel free to ask!

Much like in web browsers, the rendering of a document in WeasyPrint goes like this:

1. The HTML document is fetched and parsed into a tree of elements (like DOM).
2. CSS stylesheets (either found in the HTML or supplied by the user) are fetched and parsed.
3. The stylesheets are applied to the DOM-like tree.
4. The DOM-like tree with styles is transformed into a *formatting structure* made of rectangular boxes.
5. These boxes are *laid-out* with fixed dimensions and position onto pages.

6. For each page, the boxes are re-ordered to observe stacking rules, and are drawn on a PDF page.
7. Cairo's PDF is modified to add metadata such as bookmarks and hyperlinks.

HTML

Not much to see here. The *weasyprint.HTML* class handles step 1 and gives a tree of HTML *elements*. Although the actual API is different, this tree is conceptually the same as what web browsers call *the DOM*.

CSS

As with HTML, CSS stylesheets are parsed in the *weasyprint.CSS* class with an external library, tinycss2.

After the In addition to the actual parsing, the `weasyprint.css` and `weasyprint.css.validation` modules do some pre-processing:

- Unknown and unsupported declarations are ignored with warnings. Remaining property values are parsed in a property-specific way from raw tinycss2 tokens into a higher-level form.
- Shorthand properties are expanded. For example, `margin` becomes `margin-top`, `margin-right`, `margin-bottom` and `margin-left`.
- Hyphens in property names are replaced by underscores (`margin-top` becomes `margin_top`). This transformation is safe since none for the know (not ignored) properties have an underscore character.
- Selectors are pre-compiled with cssselect2.

The cascade

After that and still in the `weasyprint.css` package, the cascade (that's the C in CSS!) applies the stylesheets to the element tree. Selectors associate property declarations to elements. In case of conflicting declarations (different values for the same property on the same element), the one with the highest *weight* wins. Weights are based on the stylesheet's *origin*, `!important` markers, selector specificity and source order. Missing values are filled in through *inheritance* (from the parent element) or the property's *initial value*, so that every element has a *specified value* for every property.

These *specified values* are turned into *computed values* in the `weasyprint.css.computed_values` module. Keywords and lengths in various units are converted to pixels, etc. At this point the value for some properties can be represented by a single number or string, but some require more complex objects. For example, a `Dimension` object can be either an absolute length or a percentage.

The final result of the `get_all_computed_styles()` function is a big dict where keys are `(element, pseudo_element_type)` tuples, and keys are style dict objects. Elements are ElementTree elements, while the type of pseudo-element is a string for eg. `::first-line` selectors, or `None` for "normal" elements. Style dict objects are dicts mapping property names to the computed values. (The return value is not the dict itself, but a convenience `style_for()` function for accessing it.)

Formatting structure

The visual formatting model explains how *elements* (from the ElementTree tree) generate *boxes* (in the formatting structure). This is step 4 above. Boxes may have children and thus form a tree, much like elements. This tree is generally close but not identical to the ElementTree tree: some elements generate more than one box or none.

Boxes are of a lot of different kinds. For example you should not confuse *block-level boxes* and *block containers*, though *block boxes* are both. The `weasyprint.formatting_structure.boxes` module has a whole hierarchy of classes to represent all these boxes. We won't go into the details here, see the module and class docstrings.

The `weasyprint.formatting_structure.build` module takes an ElementTree tree with associated computed styles, and builds a formatting structure. It generates the right boxes for each element and ensures they conform to the models rules (eg. an inline box can not contain a block). Each box has a `style` attribute containing the style dict of computed values.

The main logic is based on the `display` property, but it can be overridden for some elements by adding a handler in the `weasyprint.html` module. This is how `<img>` and `<td colspan=3>` are currently implemented, for example.

This module is rather short as most of HTML is defined in CSS rather than in Python, in the user agent stylesheet.

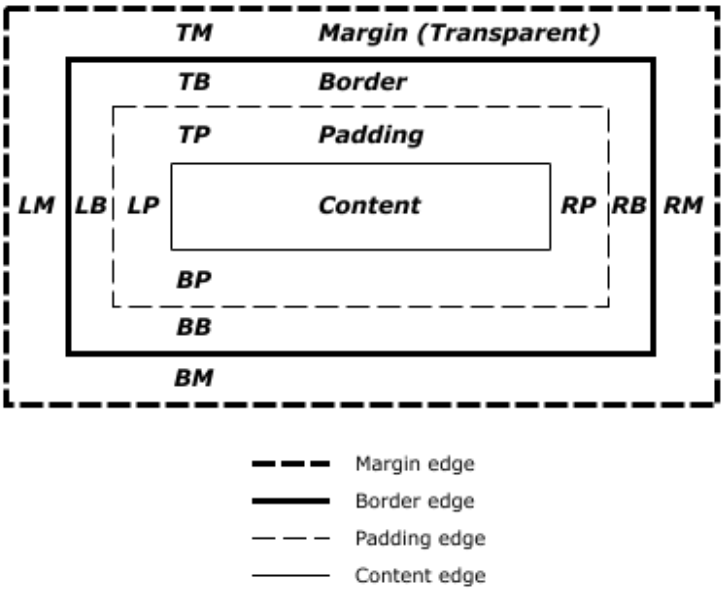
The `build_formatting_structure()` function returns the box for the root element (and, through its `children` attribute, the whole tree).

Layout

Step 5 is the layout. You could say the everything else is glue code and this is where the magic happens.

During the layout the document's content is, well, laid out on pages. This is when we decide where to do line breaks and page breaks. If a break happens inside of a box, that box is split into two (or more) boxes in the layout result.

According to the box model, each box has rectangular margin, border, padding and content areas:



While `box.style` contains computed values, the used values are set as attributes of the `Box` object itself during the layout. This include resolving percentages and especially `auto` values into absolute, pixel lengths. Once the layout done, each box has used values for margins, border width, padding of each four sides, as well as the `width` and `height` of the content area. They also have `position_x` and `position_y`, the absolute coordinates of the top-left corner of the margin box (**not** the content box) from the top-left corner of the page.[1]

Boxes also have helpers methods such as `content_box_y()` and `margin_width()` that give other metrics that can be useful in various parts of the code.

The final result of the layout is a list of `PageBox` objects.

[1] These are the coordinates *if* no CSS transform applies. Transforms change the actual location of boxes, but they are applied later during drawing and do not affect layout.

Stacking & Drawing

In step 6, the boxes are reorder by the `weasyprint.stacking` module to observe stacking rules such as the `z-index` property. The result is a tree of *stacking contexts*.

Next, each laid-out page is *drawn* onto a cairo surface. Since each box has absolute coordinates on the page from the layout step, the logic here should be minimal. If you find yourself adding a lot of logic here, maybe it should go in the layout or stacking instead.

The code lives in the `weasyprint.draw` module.

Metadata

Finally (step 7), the `weasyprint.pdf` module parses (if needed) the PDF file produced by cairo and adds metadata that cannot be added by cairo: attachments, embedded files, trim box and bleed box.

1.6 News

1.6.1 Version 43

Released on 2018-11-09.

Bug fixes:

- #726: Make empty strings clear previous values of named strings
- #729: Include tools in packaging

This version also includes the changes from unstable rc1 and rc2 versions listed below.

1.6.2 Version 43rc2

Released on 2018-11-02.

This version is experimental, don't use it in production. If you find bugs, please report them!

Bug fixes:

- #706: Fix text-indent at the beginning of a page
- #687: Allow query strings in file:// URIs
- #720: Optimize minimum size calculation of long inline elements
- #717: Display <details> tags as blocks
- #691: Don't recalculate max content widths when distributing extra space for tables
- #722: Fix bookmarks and strings set on images
- #723: Warn users when string() is not used in page margin

1.6.3 Version 43rc1

Released on 2018-10-15.

This version is experimental, don't use it in production. If you find bugs, please report them!

Dependencies:

- Python 3.4+ is now needed, Python 2.x is not supported anymore
- Cairo 1.15.4+ is now needed, but 1.10+ should work with missing features (such as links, outlines and metadata)
- Pdfrw is not needed anymore

New features:

- Beautiful website
- #579: Initial support of flexbox
- #592: Support @font-face on Windows
- #306: Add a timeout parameter to the URL fetcher functions
- #594: Split tests using modern pytest features
- #599: Make tests pass on Windows
- #604: Handle target counters and target texts
- #631: Enable counter-increment and counter-reset in page context
- #622: Allow pathlib.Path objects for HTML, CSS and Attachment classes
- #674: Add extensive installation instructions for Windows

Bug fixes:

- #558: Fix attachments
- #565, #596, #539: Fix many PDF rendering, printing and compatibility problems
- #614: Avoid crashes and endless loops caused by a Pango bug
- #662: Fix warnings and errors when generating documentation
- #666, #685: Fix many table layout rendering problems
- #680: Don't crash when there's no font available
- #662: Fix support of some align values in tables

1.6.4 Version 0.42.3

Released on 2018-03-27.

Bug fixes:

- #583: Fix floating-point number error to fix floating box layout
- #586: Don't optimize resume_at when splitting lines with trailing spaces
- #582: Fix table layout with no overflow
- #580: Fix inline box breaking function
- #576: Split replaced_min_content_width and replaced_max_content_width

- #574: Respect text direction and don't translate rtl columns twice
- #569: Get only first line's width of inline children to get linebox width

1.6.5 Version 0.42.2

Released on 2018-02-04.

Bug fixes:

- #560: Fix a couple of crashes and endless loops when breaking lines.

1.6.6 Version 0.42.1

Released on 2018-02-01.

Bug fixes:

- #566: Don't crash when using @font-config.
- #567: Fix text-indent with text-align: justify.
- #465: Fix string(*, start).
- #562: Handle named pages with pseudo-class.
- #507: Fix running headers.
- #557: Avoid infinite loops in inline_line_width.
- #555: Fix margins, borders and padding in column layouts.

1.6.7 Version 0.42

Released on 2017-12-26.

WeasyPrint is not tested with (end-of-life) Python 3.3 anymore.

This release is probably the last version of the 0.x series.

Next version may include big changes:

- end of Python 2.7 support,
- initial support of bidirectional text,
- initial support of flexbox,
- improvements for speed and memory usage.

New features:

- #532: Support relative file URIs when using CLI.

Bug fixes:

- #553: Fix slow performance for pre-formatted boxes with a lot of children.
- #409: Don't crash when rendering some tables.
- #39: Fix rendering of floats in inlines.
- #301: Split lines carefully.

- #530: Fix root when frozen with Pyinstaller.
- #534: Handle SVGs containing images embedded as data URIs.
- #360: Fix border-radius rendering problem with some PDF readers.
- #525: Fix pipenv support.
- #227: Smartly handle replaced boxes with percentage width in auto-width parents.
- #520: Don't ignore CSS @page rules that are imported by an @import rule.

1.6.8 Version 0.41

Released on 2017-10-05.

WeasyPrint now depends on pdfrw >= 0.4.

New features:

- #471: Support page marks and bleed.

Bug fixes:

- #513: Don't crash on unsupported image-resolution values.
- #506: Fix @font-face use with write_* methods.
- #500: Improve readability of _select_source function.
- #498: Use CSS prefixes as recommanded by the CSSWG.
- #441: Fix rendering problems and crashes when using @font-face.
- bb3a4db: Try to break pages after a block before trying to break inside it.
- 1d1654c: Fix and test corner cases about named pages.

Documentation:

- #508: Add missing libpangocairo dependency for Debian and Ubuntu.
- a7b17fb: Add documentation on logged rendering steps.

1.6.9 Version 0.40

Released on 2017-08-17.

WeasyPrint now depends on cssselect2 instead of cssselect and lxml.

New features:

- #57: Named pages.
- Unprefix properties, see #498.
- Add a "verbose" option logging the document generation steps.

Bug fixes:

- #483: Fix slow performance with long pre-formatted texts.
- #70: Improve speed and memory usage for long documents.
- #487: Don't crash on local() fonts with a space and no quotes.

1.6.10 Version 0.39

Released on 2017-06-24.

Bug fixes:

- Fix the use of WeasyPrint's URL fetcher with CairoSVG.

1.6.11 Version 0.38

Released on 2017-06-16.

Bug fixes:

- #477: Don't crash on font-face's src attributes with local functions.

1.6.12 Version 0.37

Released on 2017-06-15.

WeasyPrint now depends on tinycss2 instead of tinycss.

New features:

- #437: Support local links in generated PDFs.

Bug fixes:

- #412: Use a NullHandler log handler when WeasyPrint is used as a library.
- #417, #472: Don't crash on some line breaks.
- #327: Don't crash with replaced elements with height set in percentages.
- #467: Remove incorrect line breaks.
- #446: Let the logging module do the string interpolation.

1.6.13 Version 0.36

Released on 2017-02-25.

New features:

- #407: Handle ::first-letter.
- #423: Warn user about broken cairo versions.

Bug fixes:

- #411: Typos fixed in command-line help.

1.6.14 Version 0.35

Released on 2017-02-25.

Bug fixes:

- #410: Fix AssertionError in split_text_box.

1.6.15 Version 0.34

Released on 2016-12-21.

Bug fixes:

- #398: Honor the presentational_hints option for PDFs.
- #399: Avoid CairoSVG-2.0.0rc* on Python 2.
- #396: Correctly close files open by mkstemp.
- #403: Cast the number of columns into int.
- Fix multi-page multi-columns and add related tests.

1.6.16 Version 0.33

Released on 2016-11-28.

New features:

- #393: Add tests on MacOS.
- #370: Enable @font-face on MacOS.

Bug fixes:

- #389: Always update resume_at when splitting lines.
- #394: Don't build universal wheels.
- #388: Fix logic when finishing block formatting context.

1.6.17 Version 0.32

Released on 2016-11-17.

New features:

- #28: Support @font-face on Linux.
- Support CSS fonts level 3 almost entirely, including OpenType features.
- #253: Support presentational hints (optional).
- Support break-after, break-before and break-inside for pages and columns.
- #384: Major performance boost.

Bux fixes:

- #368: Respect white-space for shrink-to-fit.
- #382: Fix the preferred width for column groups.
- Handle relative boxes in column-layout boxes.

Documentation:

- Add more and more documentation about Windows installation.
- #355: Add fonts requirements for tests.

1.6.18 Version 0.31

Released on 2016-08-28.

New features:

- #124: Add MIME sniffing for images.
- #60: CSS Multi-column Layout.
- #197: Add hyphens at line breaks activated by a soft hyphen.

Bux fixes:

- #132: Fix Python 3 compatibility on Windows.

Documentation:

- #329: Add documentation about installation on Windows.

1.6.19 Version 0.30

Released on 2016-07-18.

WeasyPrint now depends on html5lib-0.999999999.

Bux fixes:

- Fix Acid2
- #325: Cutting lines is broken in page margin boxes.
- #334: Newest html5lib 0.999999999 breaks rendering.

1.6.20 Version 0.29

Released on 2016-06-17.

Bug fixes:

- #263: Don't crash with floats with percents in positions.
- #323: Fix CairoSVG 2.0 pre-release dependency in Python 2.x.

1.6.21 Version 0.28

Released on 2016-05-16.

Bug fixes:

- #189: `white-space: nowrap` still wraps on hyphens
- #305: Fix crashes on some tables
- Don't crash when transform matrix isn't invertible
- Don't crash when rendering ratio-only SVG images
- Fix margins and borders on some tables

1.6.22 Version 0.27

Released on 2016-04-08.

New features:

- #295: Support the ‘rem’ unit.
- #299: Enhance the support of SVG images.

Bug fixes:

- #307: Fix the layout of cells larger than their tables.

Documentation:

- The website is now on GitHub Pages, the documentation is on Read the Docs.
- #297: Rewrite the CSS chapter of the documentation.

1.6.23 Version 0.26

Released on 2016-01-29.

New features:

- Support the *empty-cells* attribute.
- Respect table, column and cell widths.

Bug fixes:

- #172: Unable to set table column width on tables td’s.
- #151: Table background colour bleeds beyond table cell boundaries.
- #260: TypeError: unsupported operand type(s) for +: ‘float’ and ‘str’.
- #288: Unwanted line-breaks in bold text.
- #286: AttributeError: ‘Namespace’ object has no attribute ‘attachments’.

1.6.24 Version 0.25

Released on 2015-12-17.

New features:

- Support the ‘q’ unit.

Bug fixes:

- #285: Fix a crash happening when splitting lines.
- #284: Escape parenthesis in PDF links.
- #280: Replace utf8 with utf-8 for gettext/django compatibility.
- #269: Add support for use when frozen.
- #250: Don’t crash when attachments are not available.

1.6.25 Version 0.24

Released on 2015-08-04.

New features:

- #174: Basic support for Named strings.

Bug fixes:

- #207: Draw rounded corners on replaced boxes.
- #224: Rely on the font size for rounding bug workaround.
- #31: Honor the vertical-align property in fixed-height cells.
- #202: Remove unreachable area/border at bottom of page.
- #225: Don't allow unknown units during line-height validation.
- Fix some wrong conflict resolutions for table borders with inset and outset styles.

1.6.26 Version 0.23

Released on 2014-09-16.

Bug fixes:

- #196: Use the default image sizing algorithm for images's preferred size.
- #194: Try more library aliases with `dlopen()`.
- #201: Consider `page-break-after-avoid` when pushing floats to the next page.
- #217: Avoid a crash on zero-sized background images.

Release process:

- Start testing on Python 3.4 on Travis-CI.

1.6.27 Version 0.22

Released on 2014-05-05.

New features:

- #86: Support gzip and deflate encoding in HTTP responses
- #177: Support for PDF attachments.

Bug fixes:

- #169: Fix a crash on percentage-width columns in an auto-width table.
- #168: Make `<fieldset>` a block in the user-agent stylesheet.
- #175: Fix some `dlopen()` library loading issues on OS X.
- #183: Break to the next page before a float that would overflow the page. (It might still overflow if it's bigger than the page.)
- #188: Require a recent enough version of Pyphen

Release process:

- Drop Python 3.1 support.

- Set up [Travis CI](http://travis-ci.org/) to automatically test all pushes and pull requests.
- Start testing on Python 3.4 locally. (Travis does not support 3.4 yet.)

1.6.28 Version 0.21

Released on 2014-01-11.

New features:

- Add the overflow-wrap property, allowing line breaks inside otherwise-unbreakable words. Thanks Frédérick Deslandes!
- Add the image-resolution property, allowing images to be sized proportionally to their intrinsic size at a resolution other than 96 image pixels per CSS `in` (ie. one image pixel per CSS `px`)

Bug fixes:

- #145: Fix parsing HTML from an HTTP URL on Python 3.x
- #40: Use more general hyphenation dictionnaries for specific document languages. (E.g. use `hyph_fr.dic` for `lang="fr_FR"`.)
- #26: Fix `min-width` and `max-width` on floats.
- #100: Fix a crash on trailing whitespace with `font-size: 0`
- #82: Borders on tables with `border-collapse: collapse` were sometimes drawn at an incorrect position.
- #30: Fix positioning of images with `position: absolute`.
- #118: Fix a crash when using `position: absolute` inside a `position: relative` element.
- Fix `visibility: collapse` to behave like `visibility: hidden` on elements other than table rows and table columns.
- #147 and #153: Fix dependencies to require lxml 3.0 or a more recent version. Thanks gizmonerd and Thomas Grainger!
- #152: Fix a crash on percentage-sized table cells in auto-sized tables. Thanks Johannes Duschl!

1.6.29 Version 0.20.2

Released on 2013-12-18.

- Fix #146: don't crash when drawing really small boxes with dotted/dashed borders

1.6.30 Version 0.20.1

Released on 2013-12-16.

- Depend on html5lib >= 0.99 instead of 1.0b3 to fix pip 1.4 support.
- Fix #74: don't crash on space followed by dot at line break.
- Fix #78: nicer colors for border-style: ridge/groove/inset/outset.

1.6.31 Version 0.20

Released on 2013-12-14.

- Add support for `border-radius`.
- Feature #77: Add PDF metadata from HTML.
- Feature #12: Use html5lib.
- Tables: handle percentages for column groups, columns and cells, and values for row height.
- Bug fixes:
  - Fix #84: don't crash when stylesheets are not available.
  - Fix #101: use page ids instead of page numbers in PDF bookmarks.
  - Use `logger.warning` instead of deprecated `logger.warn`.
  - Add 'font-stretch' in the 'font' shorthand.

1.6.32 Version 0.19.2

Released on 2013-06-18.

Bug fix release:

- Fix #88: `text-decoration: overline` not being drawn above the text
- Bug fix: Actually draw multiple lines when multiple values are given to `text-decoration`.
- Use the font metrics for text decoration positioning.
- Bug fix: Don't clip the border with `overflow: hidden`.
- Fix #99: Regression: JPEG images not loading with cairo 1.8.x.

1.6.33 Version 0.19.1

Released on 2013-04-30.

Bug fix release:

- Fix incorrect intrinsic width calculation leading to unnecessary line breaks in floats, tables, etc.
- Tweak border painting to look better
- Fix unnecessary page break before big tables.
- Fix table row overflowing at the bottom of the page when there are margins above the table.
- Fix `position: fixed` to actually repeat on every page.
- Fix #76: repeat `<thead>` and `<tfoot>` elements on every page, even with table border collapsing.

1.6.34 Version 0.19

Released on 2013-04-18.

- Add support for `linear-gradient()` and `radial-gradient` in background images.

- Add support for the `ex` and `ch` length units. (`1ex` is based on the font instead of being always `0.5em` as before.)
- Add experimental support for Level 4 hyphenation properties.
- Drop support for CFFI < 0.6 and cairocffi < 0.4.
- Many bug fixes, including:
- Fix #54: min/max-width/height on block-level images.
- Fix #71: Crash when parsing nested functional notation.

1.6.35 Version 0.18

Released on 2013-03-30.

- Add support for Level 3 backgrounds, including multiple background layers per element/box.
- Forward-compatibility with (future releases of) cairocffi 0.4+ and CFFI 0.6+.
- Bug fixes:
  - Avoid some unnecessary line breaks for elements sized based on their content (aka. "shrink-to-fit") such as floats and page headers.
  - Allow page breaks between empty blocks.
  - Fix #66: Resolve images' auto width from non-auto height and intrinsic ratio.
  - Fix #21: The `data:` URL scheme is case-insensitive.
  - Fix #53: Crash when backtracking for `break-before/after: avoid`.

1.6.36 Version 0.17.1

Released on 2013-03-18.

Bug fixes:

- Fix #41: GObject initialization when GDK-PixBuf is not installed.
- Fix #42: absolute URLs without a base URL (ie. document parsed from a string.)
- Fix some whitespace collapsing bugs.
- Fix absolutely-positioned elements inside inline elements.
- Fix URL escaping of image references from CSS.
- Fix #49: Division by 0 on dashed or dotted border smaller than one dot/dash.
- Fix #44: bad interaction of `page-break-before/after: avoid` and floats.

1.6.37 Version 0.17

Released on 2013-02-27.

- Added text hyphenation with the `-weasy-hyphens` property.
- When a document includes JPEG images, embed them as JPEG in the PDF output. This often results in smaller PDF file size compared to the default *deflate* compression.

- Switched to using CFFI instead of PyGTK or PyGObject-introspection.
- Layout bug fixes:
  - Correctly trim whitespace at the end of lines.
  - Fix some cases with floats within inline content.

1.6.38 Version 0.16

Released on 2012-12-13.

- Add the `zoom` parameter to *`HTML.write_pdf()`* and *`Document.write_pdf()`*
- Fix compatibility with old (and buggy) pycairo versions. WeasyPrint is now tested on 1.8.8 in addition to the latest.
- Fix layout bugs related to line trailing spaces.

1.6.39 Version 0.15

Released on 2012-10-09.

- Add a low-level API that enables painting pages individually on any cairo surface.
- **Backward-incompatible change**: remove the `HTML.get_png_pages()` method. The new low-level API covers this functionality and more.
- Add support for the `font-stretch` property.
- Add support for `@page:blank` to select blank pages.
- New Sphinx-based and improved docs
- Bug fixes:
  - Importing Pango in some PyGTK installations.
  - Layout of inline-blocks with *vertical-align: top* or *bottom*.
  - Do not repeat a block's margin-top or padding-top after a page break.
  - Performance problem with large tables split across many pages.
  - Anchors and hyperlinks areas now follow CSS transforms. Since PDF links have to be axis-aligned rectangles, the bounding box is used. This may be larger than expected with rotations that are not a multiple of 90 degrees.

1.6.40 Version 0.14

Released on 2012-08-03.

- Add a public API to choose media type used for @media. (It still defaults to `print`). Thanks Chung Lu!
- Add `--base-url` and `--resolution` to the command-line API, making it as complete as the Python one.
- Add support for the `<base href="...">` element in HTML.
- Add support for CSS outlines
- Switch to gdk-pixbuf instead of Pystacia for loading raster images.
- Bug fixes:

  - Handling of filenames and URLs on Windows
  - Unicode filenames with older version of py2cairo
  - `base_url` now behaves as expected when set to a directory name.
  - Make some tests more robust

1.6.41 Version 0.13

Released on 2012-07-23.

- Add support for PyGTK, as an alternative to PyGObject + introspection. This should make WeasyPrint easier to run on platforms that not not have packages for PyGObject 3.x yet.
- Bug fix: crash in PDF outlines for some malformed HTML documents

1.6.42 Version 0.12

Released on 2012-07-19.

- Add support for collapsed borders on tables. This is currently incompatible with repeating header and footer row groups on each page: headers and footers are treated as normal row groups on table with `border-collapse: collapse`.
- Add `url_fetcher` to the public API. This enables users to hook into WeasyPrint for fetching linked stylesheets or images, eg. to generate them on the fly without going through the network. This enables the creation of Flask-WeasyPrint.

1.6.43 Version 0.11

Released on 2012-07-04.

- Add support for floats and clear. Together with various bug fixes, this enables WeasyPrint to pass the Acid2 test! Acid2 is now part of our automated test suite.
- Add support for the width, min-width, max-width, height, min-height and max-height properties in @page. The size property is now the size of the page's containing block.
- Switch the Variable Dimension rules to the new proposal. The previous implementation was broken in many cases.
- The `image-rendering`, `transform`, `transform-origin` and `size` properties are now unprefixed. The prefixed form (eg. -weasy-size) is ignored but gives a specific warning.

1.6.44 Version 0.10

Released on 2012-06-25.

- Add `get_png_pages()` to the public API. It returns each page as a separate PNG image.
- Add a `resolution` parameter for PNG.
- Add *WeasyPrint Navigator*, a web application that shows WeasyPrint's output with clickable links. Yes, that's a browser in your browser. Start it with `python -m weasyprint.navigator`
- Add support for *vertical-align: top* and *vertical-align: bottom*

- Add support for *page-break-before: avoid* and *page-break-after: avoid*
- Bug fixes

1.6.45 Version 0.9

Released on 2012-06-04.

- Relative, absolute and fixed positioning
- Proper painting order (z-index)
- In PDF: support for internal and external hyperlinks as well as bookmarks.
- Added the `tree` parameter to the `HTML` class: accepts a parsed lxml object.
- Bug fixes, including many crashes.

Bookmarks can be controlled by the `-weasy-bookmark-level` and `-weasy-bookmark-label` properties, as described in CSS Generated Content for Paged Media Module.

The default UA stylesheet sets a matching bookmark level on all `<h1>` to `<h6>` elements.

1.6.46 Version 0.8

Released on 2012-05-07.

- Switch from cssutils to tinycss as the CSS parser.
- Switch to the new cssselect, almost all level 3 selectors are supported now.
- Support for inline blocks and inline tables
- Automatic table layout (column widths)
- Support for the `min-width`, `max-width`, `min-height` and `max-height` properties, except on table-related and page-related boxes.
- Speed improvements on big stylesheets / small documents thanks to tinycss.
- Many bug fixes

1.6.47 Version 0.7.1

Released on 2012-03-21.

Change the license from AGPL to BSD.

1.6.48 Version 0.7

Released on 2012-03-21.

- Support page breaks between table rows
- Support for the `orphans` and `widows` properties.
- Support for `page-break-inside: avoid`
- Bug fixes

Only avoiding page breaks before/after an element is still missing.

1.6.49 Version 0.6.1

Released on 2012-03-01.

Fix a packaging bug. (Remove use_2to3 in setup.py. We use the same codebase for Python 2 and 3.)

1.6.50 Version 0.6

Released on 2012-02-29.

- *Backward incompatible*: completely change the Python API. See the documentation: https://weasyprint.readthedocs.io/en/latest/tutorial.html#as-a-python-library
- *Backward incompatible*: Proper margin collapsing. This changes how blocks are rendered: adjoining margins "collapse" (their maximum is used) instead of accumulating.
- Support images in `embed` or `object` elements.
- Switch to pystacia instead of PIL for raster images
- Add compatibility with CPython 2.6 and 3.2. (Previously only 2.7 was supported)
- Many bug fixes

1.6.51 Version 0.5

Released on 2012-02-08.

- Support for the `overflow` and `clip` properties.
- Support for the `opacity` property from CSS3 Colors.
- Support for CSS 2D Transforms. These are prefixed, so you need to use `-weasy-transform` and `-weasy-transform-origin`.

1.6.52 Version 0.4

Released on 2012-02-07.

- Support `text-align: justify`, `word-spacing` and `letter-spacing`.
- Partial support for CSS3 Paged Media: page size and margin boxes with page-based counters.
- All CSS 2.1 border styles
- Fix SVG images with non-pixel units. Requires CairoSVG 0.3
- Support for `page-break-before` and `page-break-after`, except for the value `avoid`.
- Support for the `background-clip`, `background-origin` and `background-size` from CSS3 (but still with a single background per element)
- Support for the `image-rendering` from SVG. This one is prefixed, use `-weasy-image-rendering`. It only has an effect on PNG output.

1.6.53 Version 0.3.1

Released on 2011-12-14.

Compatibility with CairoSVG 0.1.2

1.6.54 Version 0.3

Released on 2011-12-13.

- **Backward-incompatible change:** the 'size' property is now prefixed (since it is in an experimental specification). Use '-weasy-size' instead.
- cssutils 0.9.8 or higher is now required.
- Support SVG images with CairoSVG
- Support generated content: the `:before` and `:after` pseudo-elements, the `content`, `quotes` and `counter-*` properties.
- Support ordered lists: all CSS 2.1 values of the `list-style-type` property.
- New user-agent stylesheet with HTML 5 elements and automatic quotes for many languages. Thanks Peter Moulder!
- Disable cssutils validation warnings, they are redundant with WeasyPrint's.
- Add `--version` to the command-line script.
- Various bug fixes

1.6.55 Version 0.2

Released on 2011-11-25.

- Support for tables.
- Support the *box-sizing* property from CSS 3 Basic User Interface
- Support all values of vertical-align except top and bottom. They are interpreted as text-top and text-bottom.
- Minor bug fixes

Tables have some limitations: Only the fixed layout and separate border model are supported. There are also no page break inside tables so a table higher than a page will overflow.

1.6.56 Version 0.1

Released on 2011-10-28.

First packaged release. Supports "simple" CSS 2.1 pages: there is no support for floats, tables, or absolute positioning. Other than that most of CSS 2.1 is supported, as well as CSS 3 Colors and Selectors.

CHAPTER 2

Authors

Development Lead:

- Simon Sapin
- Guillaume Ayoub

Contributors:

- Aarni Koskela
- Alessandro Pasotti
- Anaèle Baumgartner
- Andres Riofrio
- Aymeric Bois
- Chung Wu
- Clément Plasse
- Colin Leitner
- Florian Demmer
- Florian Mounier
- Frédérick Deslandes
- Glwadys Fayolle
- Gregory Brown
- Joe Hu
- Johan Dahlin
- Johannes Duschl
- Peter Moulder
- Pierre-Alain Mignot

- Priit Laes
- Salem Harrache
- Sergey Pikhovkin
- Smylers
- Thomas Grainger
- kaikuehne
- mbarkhau

Python Module Index

W

weasyprint, 17
weasyprint.document, 19

Index

Symbols

–base-url <URL>
command line option, 17
–version
command line option, 17
-a <file>, –attachment <file>
command line option, 17
-e <input_encoding>, –encoding <input_encoding>
command line option, 16
-f <output_format>, –format <output_format>
command line option, 16
-h, –help
command line option, 17
-m <type>, –media-type <type>
command line option, 17
-p, –presentational-hints
command line option, 17
-r <dpi>, –resolution <dpi>
command line option, 16
-s <filename_or_URL>, –stylesheet <filename_or_URL>
command line option, 16

A

add_hyperlinks() (weasyprint.document.Document method), 20
attachments (weasyprint.document.DocumentMetadata attribute), 22
authors (weasyprint.document.DocumentMetadata attribute), 21

B

bleed (weasyprint.document.Page attribute), 22

C

command line option
–base-url <URL>, 17
–version, 17
-a <file>, –attachment <file>, 17
-e <input_encoding>, –encoding <input_encoding>, 16
-f <output_format>, –format <output_format>, 16
-h, –help, 17
-m <type>, –media-type <type>, 17
-p, –presentational-hints, 17
-r <dpi>, –resolution <dpi>, 16
-s <filename_or_URL>, –stylesheet <filename_or_URL>, 16
copy() (weasyprint.document.Document method), 20
created (weasyprint.document.DocumentMetadata attribute), 21
CSS (class in weasyprint), 19

D

default_url_fetcher() (in module weasyprint), 19
description (weasyprint.document.DocumentMetadata attribute), 21
Document (class in weasyprint.document), 19
DocumentMetadata (class in weasyprint.document), 21

G

generator (weasyprint.document.DocumentMetadata attribute), 21

H

height (weasyprint.document.Page attribute), 22
HTML (class in weasyprint), 17

K

keywords (weasyprint.document.DocumentMetadata attribute), 21

M

main() (in module weasyprint.__main__), 16
make_bookmark_tree() (weasyprint.document.Document method), 20
metadata (weasyprint.document.Document attribute), 19

modified (weasyprint.document.DocumentMetadata attribute), 21

P

Page (class in weasyprint.document), 22
pages (weasyprint.document.Document attribute), 19
paint() (weasyprint.document.Page method), 22

R

render() (weasyprint.HTML method), 17
resolve_links() (weasyprint.document.Document method), 20

T

title (weasyprint.document.DocumentMetadata attribute), 21

U

url_fetcher (weasyprint.document.Document attribute), 20

W

weasyprint (module), 17
weasyprint.document (module), 19
width (weasyprint.document.Page attribute), 22
write_pdf() (weasyprint.document.Document method), 20
write_pdf() (weasyprint.HTML method), 18
write_png() (weasyprint.document.Document method), 21
write_png() (weasyprint.HTML method), 18