xhtml2pdf Documentation

Release 0.2.5

xhtml2pdf

Mar 19, 2021

CONTENTS

xhtml2pdf enables users to generate PDF documents from HTML content easily and with automated flow control such as pagination and keeping text together. The **Python module** can be used in any Python environment, including Django. The **Command line tool** is a stand-alone program that can be executed from the command line.

Contents:

CHAPTER

ONE

INSTALLATION

This is a typical Python library and is installed using pip

```
pip install xhtml2pdf
```

1.1 Requirements

Tested are Python 2.7, 3.5, 3.6, 3.7 and 3.8 at the moment. But support for Python < 3.6 will be dropped in the next release! Support for Python 3.9 is being worked on.

All additional requirements are listed in `setup.py` file and are installed automatically using the `pip install xhtml2pdf` method.

1.2 Development environment

1. If you don't have it, install `pip`, the python package installer

   ```
   sudo easy_install pip
   ```

   For more information about `pip` refer to http://www.pip-installer.org/.

2. We recommend using `virtualenv` for development. It is great to have a separate environment for each project, keeping the dependencies for multiple projects separated

   ```
   sudo pip install virtualenv
   ```

   For more information about `virtualenv` refer to http://www.virtualenv.org/

3. Create a virtualenv for the project. This can be inside the project directory, but cannot be under version control

   ```
   virtualenv --distribute xhtml2pdfenv --python=python2
   ```

4. Activate your virtualenv

   ```
   source xhtml2pdfenv/bin/activate
   ```

   Later to deactivate use

   ```
   deactivate
   ```

5. Next step will be to install/upgrade dependencies from the `requirements.txt` file

```
pip install -r requirements.txt
```

6. Run tests to check your configuration

```
python -m unittest discover tests
```

You should have a log with success status:

```
Ran 108 tests in 1.372s

OK
```

CHAPTER

TWO

USAGE

2.1 Using with Python standalone

```
from xhtml2pdf import pisa             # import python module

# Define your data
source_html = "<html><body><p>To PDF or not to PDF</p></body></html>"
output_filename = "test.pdf"

# Utility function
def convert_html_to_pdf(source_html, output_filename):
    # open output file for writing (truncated binary)
    result_file = open(output_filename, "w+b")

    # convert HTML to PDF
    pisa_status = pisa.CreatePDF(
            source_html,                # the HTML to convert
            dest=result_file)           # file handle to recieve result

    # close output file
    result_file.close()                 # close output file

    # return False on success and True on errors
    return pisa_status.err

# Main program
if __name__ == "__main__":
    pisa.showLogging()
    convert_html_to_pdf(source_html, output_filename)
```

This basic Python example will generate a test.pdf file with the text 'To PDF or not to PDF' in the top left of the page. In-memory files can be generated by using StringIO or cStringIO instead of the file open. Advanced options will be discussed later in this document.

2.2 Using xhtml2pdf in Django

To allow URL references to be resolved using Django's STATIC_URL and MEDIA_URL settings, xhtml2pdf allows users to specify a `link_callback` parameter to point to a function that converts relative URLs to absolute system paths.

```
import os
from django.conf import settings
from django.http import HttpResponse
from django.template.loader import get_template
from xhtml2pdf import pisa
from django.contrib.staticfiles import finders


    def link_callback(uri, rel):
            """
            Convert HTML URIs to absolute system paths so xhtml2pdf can access those
            resources
            """
            result = finders.find(uri)
            if result:
                    if not isinstance(result, (list, tuple)):
                            result = [result]
                    result = list(os.path.realpath(path) for path in result)
                    path=result[0]
            else:
                    sUrl = settings.STATIC_URL        # Typically /static/
                    sRoot = settings.STATIC_ROOT      # Typically /home/userX/project_
↪static/
                    mUrl = settings.MEDIA_URL         # Typically /media/
                    mRoot = settings.MEDIA_ROOT       # Typically /home/userX/project_
↪static/media/

                    if uri.startswith(mUrl):
                            path = os.path.join(mRoot, uri.replace(mUrl, ""))
                    elif uri.startswith(sUrl):
                            path = os.path.join(sRoot, uri.replace(sUrl, ""))
                    else:
                            return uri

            # make sure that file exists
            if not os.path.isfile(path):
                    raise Exception(
                            'media URI must start with %s or %s' % (sUrl, mUrl)
                    )
            return path
```

```
def render_pdf_view(request):
    template_path = 'user_printer.html'
    context = {'myvar': 'this is your template context'}
    # Create a Django response object, and specify content_type as pdf
    response = HttpResponse(content_type='application/pdf')
    response['Content-Disposition'] = 'attachment; filename="report.pdf"'
    # find the template and render it.
    template = get_template(template_path)
    html = template.render(context)
```

(continues on next page)

(continued from previous page)

```
    # create a pdf
    pisa_status = pisa.CreatePDF(
       html, dest=response, link_callback=link_callback)
    # if error then show some funy view
    if pisa_status.err:
       return HttpResponse('We had some errors <pre>' + html + '</pre>')
    return response
```

You can see in action in `demo/djangoproject` folder

2.3 Using in Command line

xhtml2pdf also provides a convenient command line tool which you can use to convert HTML files to PDF documents using the command line.

```
$ xhtml2pdf test.html
```

This basic command will convert the content of test.html to PDF and save it to test.pdf. Advanced options will be described later in this document.

The `-s` option can be used to start the default PDF viewer after the conversion:

```
$ xhtml2pdf -s test.html
```

2.4 Advanced Command line tool options

Use `xhtml2pdf --help` to get started.

2.4.1 Converting HTML data

To generate a PDF from an HTML file called `test.html` call:

```
$ xhtml2pdf -s test.html
```

The resulting PDF will be called `test.pdf` (if this file is locked e.g. by the Adobe Reader it will be called `test-0.pdf` and so on). The `-s` option takes care that the PDF will be opened directly in the Operating Systems default viewer.

To convert more than one file you may use wildcard patterns like `*` and `?`:

```
$ xhtml2pdf "test/test-*.html"
```

You may also directly access pages from the internet:

```
$ xhtml2pdf -s http://www.xhtml2pdf.com/
```

2.4.2 Using special properties

If the conversion doesn't work as expected some more informations may be usefull. You may turn on the output of warnings adding `-w` or even the debugging output by using `-d`.

Another reason could be, that the parsing failed. Consider trying the `-xhtml` and `-html` options. `xhtml2pdf` uses the HTMLT5lib parser that offers two internal parsing modes: one for HTML and one for XHTML.

When generating the HTML output `xhtml2pdf` uses an internal default CSS definition (otherwise all tags would appear with no diffences). To get an impression of how this one looks like start `xhtml2pdf` like this:

```
$ xhtml2pdf --css-dump > xhtml2pdf-default.css
```

The CSS will be dumped into the file `xhtml2pdf-default.css`. You may modify this or even take a totaly self defined one and hand it in by using the `-css` option, e.g.:

```
$ xhtml2pdf --css=xhtml2pdf-default.css test.html
```

CHAPTER

THREE

WORKING WITH HTML

3.1 PDF vs. HTML

Before we discuss how to define page layouts with xhtml2pdf style sheets, it helps to understand some of the inherent differences between PDF and HTML.

PDF is specifically designed around pages of a specific width and height. PDF page elements (such as paragraphs, tables and images) are positioned at absolute (X,Y) coordinates.

Note: While true PDF files use (0,0) to denote the bottom left of a page, xhtml2pdf uses (0,0) to denote the top left of a page, partly to maintain similarity with the HTML coordinate system.

HTML, by itself, does not have the concept of pagination or of pages with a specific width and height. An HTML page can be as wide as your browser width (and even wider), and it can be as long as the page content itself. HTML page elements are positioned in relationship to each other and may change when the browser window is resized.

In order to bridge the differences between HTML and PDFs, xhtml2pdf makes use of the concept of **Pages** and **Frames**. Pages define the size, orientation and margins of pages. Frames are rectangular regions with in each page.

The **Frame location is specified in absolute (X,Y) coordinates**, while the **Frame content is used to flow HTML content using the relative positioning rules of HTML**. This is the essence from which the power of xhtml2pdf stems.

HTML content will start printing in the first available content frame. Once the first frame is filled up, the content will continue to print in the next frame, and the next, and so on, until the entire HTML content has been printed.

3.2 Defining Page Layouts

xhtml2pdf facilitates the conversion of HTML content into a PDF document by flowing the continuous HTML content through one or more pages using Pages and Frames. A page represents a page layout within a PDF document. A Frame represents a rectangular region within a page through which the HTML content will flow.

3.2.1 Pages

The **@page** object defines a **Page template** by defining the properties of a page, such as page type (letter or A4), page orientation (portrait or landscape), and page margins. The @page definition follows the style sheet convention of ordinary CSS style sheets:

```
<style>
    @page {
        size: letter landscape;
        margin: 2cm;
    }
</style>
```

3.2.2 Frames

The **@frame** object defines the position and size of rectangular region within a page. A @page object can hold one or more @frame objects. The @frame definition follows the style sheet convention of ordinary CSS style sheets:

Here's a definition of a page template with one Content Frame. It makes use of the Letter page size of 612pt x 792pt.

```
<style>
    @page {
        size: letter portrait;
        @frame content_frame {
            left: 50pt;
            width: 512pt;
            top: 50pt;
            height: 692pt;
            -pdf-frame-border: 1;     /* for debugging the layout */
        }
    }
</style>
```

This will result in the following page layout:

```
+-page----------------+
|                     |
|  +-content_frame-+  |
|  |               |  |
|  |               |  |
|  |               |  |
|  |               |  |
|  +---------------+  |
|                     |
+---------------------+
```

Note: To visualize the page layout you are defining, add the `-pdf-frame-border: 1;` property to your each of your @frame objects.

3.2.3 Static frames vs Content frames

xhtml2pdf uses the concept of **Static Frames** to define content that remains the same across different pages (like headers and footers), and uses **Content Frames** to position the to-be-converted HTML content.

Static Frames are defined through use of the @frame property `-pdf-frame-content`. Regular HTML content will not flow through Static Frames.

Content Frames are @frame objects without this property defined. Regular HTML content will flow through Content Frames.

3.2.4 Example with 2 Static Frames and 1 Content Frame

```
<html>
<head>
<style>
    @page {
        size: a4 portrait;
        @frame header_frame {           /* Static Frame */
            -pdf-frame-content: header_content;
            left: 50pt; width: 512pt; top: 50pt; height: 40pt;
        }
        @frame content_frame {          /* Content Frame */
            left: 50pt; width: 512pt; top: 90pt; height: 632pt;
        }
        @frame footer_frame {           /* Another static Frame */
            -pdf-frame-content: footer_content;
            left: 50pt; width: 512pt; top: 772pt; height: 20pt;
        }
    }
</style>
</head>

<body>
    <!-- Content for Static Frame 'header_frame' -->
    <div id="header_content">Lyrics-R-Us</div>

    <!-- Content for Static Frame 'footer_frame' -->
    <div id="footer_content">(c) - page <pdf:pagenumber>
        of <pdf:pagecount>
    </div>

    <!-- HTML Content -->
    To PDF or not to PDF
</body>
</html>
```

In the example above, the vendor-specific tags `<pdf:pagenumber>` and `<pdf:pagecount>` are used to display page numbers and the total page count. This example will produce the following PDF Document:

```
+-page------------------+
| +-header_frame------+ |
| | Lyrics-R-Us       | |
| +-------------------+ |
| +-content_frame-----+ |
| | To PDF or not to  | |
```

(continues on next page)

(continued from previous page)

```
| | PDF                | |
| |                    | |
| |                    | |
| +--------------------+ |
| +-footer_frame------+ |
| | (c) - page 1 of 1 | |
| +--------------------+ |
+------------------------+
```

```
# Developer's note:
# To avoid a problem where duplicate numbers are printed,
# make sure that these tags are immediately followed by a newline.
```

3.2.5 Flowing HTML content through Content Frames

Content frames are used to position the HTML content across multiple pages. HTML content will start printing in the first available Content Frame. Once the first frame is filled up, the content will continue to print in the next frame, and the next, and so on, until the entire HTML content has been printed. This concept is illustrated by the example below.

3.2.6 Example page template with a header, two columns, and a footer

```
<html>
<head>
<style>
    @page {
        size: letter portrait;
        @frame header_frame {           /* Static frame */
            -pdf-frame-content: header_content;
            left: 50pt; width: 512pt; top: 50pt; height: 40pt;
        }
        @frame col1_frame {             /* Content frame 1 */
            left: 44pt; width: 245pt; top: 90pt; height: 632pt;
        }
        @frame col2_frame {             /* Content frame 2 */
            left: 323pt; width: 245pt; top: 90pt; height: 632pt;
        }
        @frame footer_frame {           /* Static frame */
            -pdf-frame-content: footer_content;
            left: 50pt; width: 512pt; top: 772pt; height: 20pt;
        }
    }
</style>
<head>
<body>
    <div id="header_content">Lyrics-R-Us</div>
    <div id="footer_content">(c) - page <pdf:pagenumber>
        of <pdf:pagecount>
    </div>
    <p>Old MacDonald had a farm. EIEIO.</p>
    <p>And on that farm he had a cow. EIEIO.</p>
    <p>With a moo-moo here, and a moo-moo there.</p>
    <p>Here a moo, there a moo, everywhere a moo-moo.</p>
```

(continues on next page)

(continued from previous page)

```
</body>
</html>
```

The HTML content will flow from Page1.Col1 to Page1.Col2 to Page2.Col1, etc. Here's what the resulting PDF document could look like:

```
+--------------------------------+    +--------------------------------+
| Lyrics-R-Us                    |    | Lyrics-R-Us                    |
|                                |    |                                |
| Old MacDonald   farm he had a  |    | a moo-moo       everywhere a   |
| had a farm.     cow. EIEIO.    |    | there.          moo-moo.       |
| EIEIO.          With a moo-    |    | Here a moo,                    |
| and on that     moo here, and  |    | there a moo,                   |
|                                |    |                                |
| (c) - page 1 of 2              |    | (c) - page 2 of 2              |
+--------------------------------+    +--------------------------------+
```

CHAPTER

FOUR

ADVANCED CONCEPTS

4.1 Keeping text and tables together

You can prevent a block of text from being split across separate frames through the use of the vendor-specific `-pdf-keep-with-next` property.

Here's an example where paragraphs and tables are kept together until a 'separator paragraph' appears in the HTML content flow.

```
<style>
    table { -pdf-keep-with-next: true; }
    p { margin: 0; -pdf-keep-with-next: true; }
    p.separator { -pdf-keep-with-next: false; font-size: 6pt; }
</style>
  ...
<body>
    <p>Keep these lines</p>
    <table><tr><td>And this table</td></tr></table>
    <p>together in one frame</p>

    <p class="separator"> <p>

    <p>Keep these sets of lines</p>
    <p>may appear in a different frame</p>
    <p class="separator"> <p>
</body>
```

4.2 Named Page templates

Page templates can be named by providing the name after the @page keyword

```
@page my_page {
    margin: 40pt;
}
```

4.3 Switching between multiple Page templates

PDF documents sometimes requires a different page layout across different sections of the document. xhtml2pdf allows you to define multiple @page templates and a way to switch between them using the vendor-specific tag `<pdf:nexttemplate>`.

As an illustration, consider the following example for a title page with large 5cm margins and regular pages with regular 2cm margins.

```
<html>
<head>
<style>
    @page title_template { margin: 5cm; }
    @page regular_template { margin: 2cm; }
</style>
</head>

<body>
    <h1>Title Page</h1>
    This is a title page with a large 5cm margin.

    <!-- switch page templates -->
    <pdf:nexttemplate name="regular_template" />

    <h1>Chapter 1</h1>
    This is a regular page with a regular 2cm margin.
</body>
</html>
```

CHAPTER

FIVE

HTTPS OPTION

5.1 Using in python

Basically you need to set httpConfig before call pisa.CreatePDF

```
from xhtml2pdf.config.httpconfig import httpConfig
httpConfig.save_keys('nosslcheck', True)
pisaStatus = pisa.CreatePDF(
        sourceHtml,
        dest=resultFile)
```

Other way is setting as a dict

```
import ssl
from xhtml2pdf.config.httpconfig import httpConfig
httpConfig['context']=ssl._create_unverified_context()
```

In this way you can insert arbitrary httplib.HTTPSConnection parameters, for more information see:

- python2 : https://docs.python.org/2/library/httplib.html#httplib.HTTPSConnection
- python3 : https://docs.python.org/3.4/library/http.client.html#http.client.HTTPSConnection

5.2 Using in shell

So you can call to xhtml2pdf passing httplib parameters with something like:

```
xhtml2pdf --http_nosslcheck https://domain yourfile.pdf
```

the `--http_nosslcheck` is a special argument that help to disable the ssl certificate check.

See http.client.HTTPSConnection documentation for this parameters

Those are the available options:

- http_nosslcheck
- http_key_file
- http_cert_file
- http_source_address
- http_timeout

available settings

- http_key_file
- http_cert_file
- http_source_address
- http_timeout

CHAPTER

SIX

HOW TO RUN TESTS

This file describes how people should run the various test suites included with xhtml2pdf.

6.1 Unit tests

Running unit tests should be pretty intuitive to most python developers. xhtml2pdf uses the standard library's unittest to write tests. As such, the following command should "just work":

```
python -m unittest discover tests
```

A few extra bells and whistles are available. Specifically, a .coveragerc file is included with this project, therefore running coverage reports should give you immediately useful information:

```
pip install coverage
coverage run -m unittest discover tests
coverage run -a testrender/testrender.py
coverage report
```

The coverage percentage is currently pretty low - feel free to add extra tests!

6.2 Functional tests

xhtml2pdf ships with a functional tests suite. To see it in action, run the following commands from the xhtml2pdf directory:

```
python testrender/testrender.py
x-www-browser testrender/output/index.html
```

The suite renders a set of templates to pdf, then uses imagemagick (available on most unix-like systems) to convert the pdfs to png images, and finally creates a image of differences between the generated image and a reference image.

Image sets with a "difference score" of more than 0 are highlighted in red - this means the rendering library produced a bad result.

Font rendering is a very tricky business. As such, the functional suite often creates "ghost differences" for some font renderings (the images look perfect for a human eye, but the computer gives them a bad score anyway).

To solve theses, you should try regenerating reference images on your particular system, so that the exact mechanism used on your platform are used in both cases:

```
python testrender/testrender.py --create-reference local_reference
python testrender/testrender.py --ref-dir local_reference
x-www-browser testrender/output/index.html
```

You can now happily hack away at the library, without any ghost images.

CHAPTER

SEVEN

REFERENCE

7.1 Supported @page properties and values

Valid @page properties:

```
background-image
size
margin, margin-bottom, margin-left, margin-right, margin-top
```

Valid size syntax and values:

```
Syntax: @page { size: <type> <orientation>; }

Where <type> is one of:
a0 .. a6
b0 .. b6
elevenseventeen
legal
letter

And <orientation> is one of:
landscape
portrait

Defaults to:
size: a4 portrait;
```

7.2 Supported @frame properties:

Valid @frame properties.

```
bottom, top, height
left, right, width
margin, margin-bottom, margin-left, margin-right, margin-top
```

To avoid unexpected results, please only specify two out of three bottom/top/height properties, and two out of three left/right/width properties per @frame object.

7.3 Supported CSS properties

xhtml2pdf supports the following standard CSS properties

```
background-color
border-bottom-color, border-bottom-style, border-bottom-width
border-left-color, border-left-style, border-left-width
border-right-color, border-right-style, border-right-width
border-top-color, border-top-style, border-top-width
colordisplay
font-family, font-size, font-style, font-weight
height
line-height, list-style-type
margin-bottom, margin-left, margin-right, margin-top
padding-bottom, padding-left, padding-right, padding-top
page-break-after, page-break-before
size
text-align, text-decoration, text-indent
vertical-align
white-space
width
zoom
```

xhtml2pdf adds the following vendor-specific properties:

```
-pdf-frame-border
-pdf-frame-break
-pdf-frame-content
-pdf-keep-with-next
-pdf-next-page
-pdf-outline
-pdf-outline-level
-pdf-outline-open
-pdf-page-break
```

7.4 @page background-image

To add a watermark to the PDF, use the `background-image` property to specify a background image

```
@page {
    background-image: url('/path/to/pdf-background.jpg');
}
```

7.5 Create PDF

The main function of xhtml2pdf is called CreatePDF(). It offers the following arguments in this order:

- **src**: The source to be parsed. This can be a file handle or a `String` - or even better - a `Unicode` object.
- **dest**: The destination for the resulting PDF. This has to be a file object wich will not be closed by `CreatePDF`. (XXX allow file name?)
- **path**: The original file path or URL. This is needed to calculate relative paths of images and style sheets. (XXX calculate automatically from src?)

- **link_callback**: Handler for special file paths (see below).
- **debug**: ** DEPRECATED **
- **show_error_as_pdf**: Boolean that indicates that the errors will be dumped into a PDF. This is usefull if that is the only way to show the errors like in simple web applications.
- **default_css**: Here you can pass a default CSS definition in as a `String`. If set to `None` the predefined CSS of xhtml2pdf is used.
- **xhtml**: Boolean to force parsing the source as XHTML. By default the HTML5 parser tries to guess this.
- **encoding**: The encoding name of the source. By default this is guessed by the HTML5 parser. But HTML with no meta information this may not work an then this argument is helpfull.

7.6 Link callback

Images, backgrounds and stylesheets are loaded form an HTML document. Normally `xhtml2pdf` expects these files to be found on the local drive. They may also be referenced relative to the original document. But the programmer might want to load form different kind of sources like the Internet via HTTP requests or from a database or anything else. Therefore you may define a `link_callback` that handles these requests.

XXX

7.7 Web applications

XXX

7.8 Defaults

- The name of the first layout template is `body`, but you better leave the name empty for defining the default template (XXX May be changed in the future!)

7.9 Fonts

By default there is just a certain set of fonts available for PDF. Here is the complete list - and their repective alias names - `xhtml2pdf` knows by default (the names are not case sensitive):

- **Times-Roman**: Times New Roman, Times, Georgia, serif
- **Helvetica**: Arial, Verdana, Geneva, sansserif, sans
- **Courier**: Courier New, monospace, monospaced, mono
- **ZapfDingbats**
- **Symbol**

7.10 Asian Fonts Support

Now some Asian fonts are available by default for PDF. The names are not case sensitive.

Simplified Chinese:

- **STSong-Light**

Traditional Chinese:

- **MSung-Light**

Japanese:

- **HeiseiMin-W3**
- **HeiseiKakuGo-W5**

Korean:

- **HYSMyeongJo-Medium**
- **HYGothic-Medium**

Just use them in the `font-family` property in your CSS definition.

```
<style>
p { font-family: STSong-Light }
</style>
```

If you need another font, you may have a look at the "Using Custom Fonts" section.

7.11 Arabic / Hebrew / Persian etc. Fonts Support

If you are using a language with right-to-left writing you need to specify the language name in the `<pdf:language name=""/>` custom tag. This is necessary to ensure the correct direction (right to left).

The following attributes for right-to-left languages are supported and tested:

- `name="arabic"`
- `name="hebrew"`
- `name="persian"`
- `name="urdu"`
- `name="pashto"`
- `name="sindhi"`

Usage example:

```
<pdf:language name="arabic"/>

<p>Some Arabic text here</p>
<p>Some English text here</p>
```

The Arabic letters will render from right to left, while all other Latin letters will keep their left-to-right direction.

Warning: Right now it seems like right-to-left support isn't working while using a default font-family like `p { font-family: Times-Roman }`. We're working on fixing this. However, it works by using the `@font-face` tag in the CSS definition and defining a custom font. Therefore you need the specified font file. "MarkaziText" for example seems to work. It can be downloaded for free here: https://fonts.google.com/specimen/Markazi+Text Other fonts might work as well but haven't been tested.

```
<style>
    @font-face {font-family: MyRightToLeftFont; src: url('path\to\the\font\file\
↪MarkaziText-Regular.ttf')}

    p { font-family: MyRightToLeftFont }
</style>
```

7.12 Using Custom Fonts

You may also embed a new font by using the `@font-face` keyword in CSS like this:

```
@font-face {
  font-family: Example, "Example Font";
  src: url('example.ttf');
}
```

The `font-family` property defines the names under which the embedded font will be known. `src` defines the place of the fonts source file. This can be a TrueType font or a Postscript font. The file name of the first has to end with `.ttf` the latter with one of `.pfb` or `.afm`. For Postscript fonts pass just one filename like `<name>.afm` or `<name>.pfb`, the missing one will be calculated automatically.

To define other shapes you can do the following:

```
/* Normal */
@font-face {
   font-family: DejaMono;
   src: url('font/DejaVuSansMono.ttf');
}

/* Bold */
@font-face {
   font-family: DejaMono;
   src: url('font/DejaVuSansMono-Bold.ttf');
   font-weight: bold;
}

/* Italic */
@font-face {
   font-family: DejaMono;
   src: url('font/DejaVuSansMono-Oblique.ttf');
   font-style: italic;
}

/* Bold and italic */
@font-face {
   font-family: DejaMono;
   src: url('font/DejaVuSansMono-BoldOblique.ttf');
   font-weight: bold;
```

(continues on next page)

(continued from previous page)

```
    font-style: italic;
}
```

7.13 Using TFF files with the same face-name

In specific situations we have to use .ttf files with the same face name, but working with these kind of files makes us deal with some issues. To avoid it you have to add # at the beginning of the `font-family name`. Please check the following example:

```
/* put in quotes and add # at the beginning */
@font-face {
    font-family: '#MY';
    src: url('font/Microsoft YaHei.ttf')
}
```

7.14 Outlines/ Bookmarks

PDF supports outlines (Adobe calls them "bookmarks"). By default `xhtml2pdf` defines the `<h1>` to `<h6>` tags to be shown in the outline. But you can specify exactly for every tag which outline behaviour it should have. Therefore you may want to use the following vendor specific styles:

- **`-pdf-outline`** set it to "true" if the block element should appear in the outline
- **`-pdf-outline-level`** set the value starting with "0" for the level on which the outline should appear. Missing predecessors are inserted automatically with the same name as the current outline
- `-pdf-outline-open` set to "true" if the outline should be shown uncollapsed

Example:

```
h1 {
  -pdf-outline: true;  -pdf-level: 0;
  -pdf-open: false;
}
```

7.15 Table of Contents

It is possible to automatically generate a Table of Contents (TOC) with `xhtml2pdf`. By default all headings from `<h1>` to `<h6>` will be inserted into that TOC. But you may change that behaviour by setting the CSS property `-pdf-outline` to `true` or `false`. To generate the TOC simply insert `<pdf:toc />` into your document. You then may modify the look of it by defining styles for the `pdf:toc` tag and the classes `pdftoc.pdftoclevel0` to `pdftoc.pdftoclevel5`. Here is a simple example for a nice looking CSS:

```
pdftoc {
    color: #666;
}
pdftoc.pdftoclevel0 {
    font-weight: bold;
    margin-top: 0.5em;
```

(continues on next page)

(continued from previous page)

```
}
pdftoc.pdftoclevel1 {
    margin-left: 1em;
}
pdftoc.pdftoclevel2 {
    margin-left: 2em;
    font-style: italic;
}
```

7.16 Tables

Tables are supported but may behave a little different to the way you might expect them to do. These restriction are due to the underlying table mechanism of ReportLab.

- The main restriction is that table cells that are longer than one page lead to an error
- Tables can not float left or right and can not be inlined

7.17 Long cells

`xhtml2pdf` is not able to split table cells that are larger than the available space. To work around it you may define what should happen in this case. The `-pdf-keep-in-frame-mode` can be one of: "error", "overflow", "shrink", "truncate", where "shrink" is the default value.

```
table {    -pdf-keep-in-frame-mode: shrink;}
```

7.18 Cell widths

The table renderer is not able to adjust the width of the table automatically. Therefore you should explicitly set the width of the table and to the table rows or cells.

7.19 Headers

It is possible to repeat table rows if a page break occurs within a table. The number of repeated rows is passed in the property `repeat`. Example:

```
<table repeat="1">
  <tr><th>Column 1</th><th>...</th></tr>
  ...
</table>
```

7.20 Borders

Borders are supported. Use corresponding CSS styles.

7.21 Images

7.22 Size

By default JPG images are supported. If the Python Imaging Library (PIL) is installed the file types supported by it are available too. As mapping pixels to points is not trivial the images may appear bigger in the PDF as in the browser. To adjust this you may want to use the `zoom` style. Here is a small example:

```
img { zoom: 80%; }
```

7.23 Position/ floating

Since Reportlab Toolkit does not yet support the use of images within paragraphs, images are always rendered in a seperate paragraph. Therefore floating is not available yet.

7.24 Barcodes

You can embed barcodes automatically in a document. Various barcode formats are supported through the `type` property. If you want the original barcode text to be appeared on the document, simply add `humanreadable="1"`, otherwise simply omit this property. Some barcode formats have a checksum as an option and it will be on by default, set `checksum="0"` to override. Alignment is achieved through `align` property and available values are any of `"baseline", "top", "middle", "bottom"` whereas default is `baseline`. Finally, bar width and height can be controlled through `barwidth` and `barheight` properties respectively.

```
<pdf:barcode value="BARCODE TEXT COMES HERE" type="code128" humanreadable="1" align=
↪"right" />
```

7.25 Custom Tags

`xhtml2pdf` provides some custom tags. They are all prefixed by the namespace identifier `pdf:`. As the HTML5 parser used by xhtml2pdf does not know about these specific tags it may be confused if they are without a block. To avoid problems you may condsider sourrounding them by `<div>` tags, like this:

```
<div>
   <pdf:toc />
</div>
```

7.26 Tag-Definitions

7.26.1 pdf:barcode

Creates a barcode.

7.26.2 pdf:pagenumber

Prints current page number. The argument "example" defines the space the page number will require e.g. "00".

7.26.3 pdf:pagecount

Prints total page count.

7.26.4 pdf:nexttemplate

Defines the template to be used on the next page. The name of the template is passed via the `name` property and refers to a `@page templateName` style definition:

```
<pdf:nexttemplate name="templateName">
```

7.26.5 pdf:nextpage

Create a new page after this position.

7.26.6 pdf:nextframe

Jump to next unused frame on the same page or to the first on a new page. You may not jump to a named frame.

7.26.7 pdf:spacer

Creates an object of a specific size.

7.26.8 pdf:toc

Creates a Table of Contents.

7.26.9 pdf:language

Used for languages with right-to-left writing like Arabic, Hebrew, Persion etc. Right-to-left writing can be defined by passing the name via the `name=""` property.

```
<pdf:language name="arabic"/>
```

CHAPTER

EIGHT

RELEASE NOTES

8.1 Versions >= 0.2

8.1.1 0.2.5

Released: 2020-10-08

New

- Added Asian fonts support (Simplified Chinese, Traditional Chinese, Japanese & Korean) #353
- Added support for right-to-left writings like Arabic, Hebrew, Persian, Pashto, Urdu and Sindhi. Simply include for example `<pdf:language name="arabic"/>` #494

Improvements

- CSS property `letter-spacing` now supports float values and relative & absolute units like `cm`, `in`, `em`, `%` etc. #490
- Added unit tests for Asian and right-to-left fonts. #520

Bug-Fixes

- `@frame` properties like `width`, `right`, `bottom` etc. are now correctly calculated depending on the page orientation and size #499
- Fixed support for multiple fonts and unicode #492
- Fixed an encoding issue with html5lib #468
- Fixed a problem with the `border` property in `h1` to `h6` heading tags #466 #495
- Fixed compability with ReportLab 3.5.X #404 #463
- Removed default background-image when no background-image is defined #484
- Fixed an issue with different font type that have the same name #381
- Fixed a bug that prevented support for Python 3.X #513
- testrender test: fixed transparences and included new reference files, (now all tests pass in Travis CI without –failed parameter) #502
- `0.0` as value for a CSS property now acts the same way as `0` and `None` #516

Deprecation

- Removed `i` and `inch` as unofficial synonyms for the `in` unit #516

Documentation

- Added new section about Asian font support #505 #520
- Added new section about support for right-to-left writings #520
- Readme.rst file was updated #507 #512
- Added missing changelog entries for earlier releases #478

Cleanup

- Replaced deprecated `base64.encodestring` with `base64.encodebytes` #472
- Replaced deprecated `log.warn()` with `log.warning()` #509
- Dropped dependency of nose (outdated & unmaintained) in favor of unittest, which is included in the Python standard library #520
- Removed the old nose tests and replaced them with unittest #520
- Removed unlicensed .tff font files in our tests folder and replaced them with open source fonts #520
- Travis CI and AppVeyor are now testing both against the same ReportLab versions (3.3 to 3.5.X) #520

Thanks to the following people on GitHub for contributing to this release:
ezawadzki, *fbernhart*, *KirilNN*, *luisza*, *Mark-Hetherington*, *parthjoshi2007*, *pedroszg*, *silvio-dp*, *sj175*, *tirkarthi* and *z4c*

8.1.2 0.2.4

Released: 2020-01-18

New

- Add `em` unit support

Improvements

- Added testing for Python 3.7 and 3.8
- Added support for urllib in Python 2 and Python 3

Bug-Fixes

- Fixed cgi escape util on setup version
- Fixed width assignation on fragments
- Repaired base64 unscaped string
- Fixed urlparse when urls has parameters
- Fixed i_rgbcolor support

Documentation

- Updated `link_callback` documentation
- Stylized code lines in documentation

8.1.3 0.2.3

Released: 2018-09-14

Changes were not documented

8.1.4 0.2.2

Released: 2018-04-16

Changes were not documented

8.1.5 0.2.1

Released: 2018-02-16

New

- Added support for Python 3.8

Improvements

- Improved table tests

Bug-Fixes

- Forced html5lib to 1.0.1 (old versions of html5lib are not in pip)
- Allow for URI-escaped strings in base64 data

Cleanup

- Removed the dependency on httplib2

8.1.6 0.2

Released: 2018-02-15

New

- Support for a new `@page` property: `background-image`

Improvements

- Improved Python 3 support
- Included new `httplib` options

Bug-Fixes

- Fix for transparent images in Python 3

Deprecation

- Removed support for Python 2.3

Documentation

- Readthedocs integration
- Updated Django demo site

Cleanup

- PEP8 improvements and code cleanups
- Dropped the `turbogears` module

Thanks to the following people on GitHub for contributing to this release:
andreyfedoseev, *browniebroke*, *flupzor* and *luisza*

8.1.7 0.2beta1

Released: 2016-11-30

Changes were not documented

8.2 Versions >= 0.1, < 0.2

8.2.1 0.1beta3

Released: 2016-08-16

Changes were not documented

8.2.2 0.1beta2

Released: 2016-08-01

Changes were not documented

8.2.3 0.1beta1

Released: 2016-06-05

Changes were not documented

8.2.4 0.1alpha4

Released: 2016-05-18

- Removed PyPy support
- Avoid exceptions likely to occur systematic to how narrow a text column is #309 - thanks *jkDesignDE*
- Improved tests for tables #305 - thanks *taddeimania*
- Fix broken empty PDFs in Python2 #301 - thanks *citizen-stig*
- Unknown page sizes now raise an exception #71 - thanks *benjaoming*
- Unorderable types caused by duplicate CSS selectors / rules #69 - thanks *benjaoming*
- Allow empty page definition with no space after @page - #88 - thanks *benjaoming*
- Error when in addFromFile using file-like object #245 - thanks *benjaoming*
- Python 3: Bad table formatting with empty columns #279 - thanks *citizen-stig and benjaoming*
- Removed paragraph2.py, unused ghost file since the beginning of the project #289 - thanks *citizen-stig*
- Catch-all exceptions removed in a lot of places, not quite done #290 - thanks *benjaoming*

8.2.5 0.1alpha3

Released: 2016-05-01

- Improved six usage, simplifies codebase #288 - thanks *citizen-stig*
- Removed mutable types as default args #287 - thanks *citizen-stig*
- Fix “hangs forever on simple input” #209
- Base64 inline <img> works now #281

8.2.6 0.1alpha2

Released: 2016-04-14

- Fixed: AttributeError: ‘bytes’ object has no attribute ‘encode’ #265
- Improved tests, added code coverage

8.2.7 0.1alpha1

Released: 2016-01-20

This major version bump signals that we have added Python 3 support. Other than that, the project remains largely unchanged.

- Python 3 support
- Cleaning up codebase

- Github and documentation modernizations

8.3 Versions < 0.1

8.3.1 0.0.6

Released: 2014-04-27

- get css backgrounds and fonts relative to the css file path
- fix CSS parser breaking on "@media screen and ..." (issue 132)

8.3.2 0.0.5

Released: 2013-03-25

- Switched dependency to Pillow instead of PIL.
- Converted the docs to rst (thanks tomscytale!)
- Huge performance improvements (thanks Andrea Bravetti!)
- Bugfixes.

8.3.3 0.0.4

Released: 2012-05-23

- Added a <pdf:pagecount/> tag to write the total number of pages.
- The <pdf:barcode/> tag now accepts a fontsize argument for the human-readable font.
- Various bugfixes and enhancements

8.3.4 0.0.3

Released: 2011-06-19

Changes were not documented

8.3.5 0.0.2

Released: 2011-05-27

Changes were not documented

8.3.6 0.0.1

Released: 2011-05-20

Changes were not documented

8.3.7 0.0.0

Released: 2011-05-19

Changes were not documented

8.4 Legacy Versions

The following changelog entries were relevant before the maintainer change.

“I would like to thank the people mentioned in brackets in this change log very much for their help and support!” - Dirk

Version 3.0.33, 2010-06-16

- NEW: Changed license to Apache License 2.0, now completely Open Source without any charging. Feel free to continue or for this project.
- FIX: Empty cells now collapse

Version 3.0.32, 2009-05-08

- NEW: New command line option ‘–base’ to specify base path if input comes via STDIN
- FIX: The ‘keep in frame’ feature for tables did not work inside of static frames (Arun Shanker Prasad)
- FIX: Small typos

Version 3.0.31, 2009-05-04

- NEW: Support for Style “list-style-image”, also supports “zoom”
- NEW: Temporary files internally are written to disk if they exceed a certain size
- NEW: Font names can now also read from external URL
- UPD: Modified pdfjoiner.py demo
- FIX: Custom font image problem still appeared
- FIX: Single image in a block issue
- FIX: Randomly used wrong images is fixed using a workaround for the bug in Reportlab _digester routine

- FIX: Empty tables error (Davide Moro)
- FIX: Fallback to urllib2 if httpdlib fails

Version 3.0.30, 2009-03-27

- NEW: Default CSS now hides content of <noscript>
- UPD: Better whitespace handling in RL Paragraph
- FIX: Fixed RL Paragraph.split to work with autoleading and images
- FIX: Small bug fix for show_error_as_pdf
- FIX: Demos used os.startfile which is not supported on non Windows OSes
- FIX: Table available height threw exceptions
- FIX: Switched from urllib2 to httplib for loading external sources
- FIX: Correct homepage and download page in setup.py
- FIX: Paragraphs in lists repeated the bullet
- FIX: Tables now support -pdf-keep-with-next
- FIX: TOC bug fixed
- FIX: Add missing table columns to avoid error in Reportlab table
- FIX: Fix for background images sizing
- FIX: Empty documents now create one blank page
- FIX: Imported fonts caused an error if used together with images

Version 3.0.29, 2008-12-01

- NEW: Warning if Reportlab 2.2 is not installed
- UPD: Better support for named colors
- UPD: Modifed frame handling to better support relative values
- FIX: Splitting paragraph threw errors some times; also had problems with line breaks on the second page, fix for RL 2.2 paragraph was needed
- FIX: Added margins to <blockquote> default CSS
- FIX: Inline images in static frames did not work
- FIX: Link anchors and non internal fonts caused a strange error

Version 3.0.28, 2008-11-21

- NEW: Requires Reportlab 2.2 now!
- NEW: Background colors for inline elements like
- NEW: Inline images and left and right aligned images implemented
- NEW: Possibility to handle table cells that are to large via CSS option -pdf-keep-in-frame-mode
- NEW: Option “–system” for command line tool to dump system version infos
- NEW: CSS attribute -pdf-line-spacing for fix space between lines
- NEW: Creation and handling of data URI with base64 encoding (others to come)
- NEW: New general file loader that is also able to load remote data and data URI

- NEW: PDF Joiner to concatenate many PDF and pisa documents
- NEW: Page backgrounds can now be images or PDF
- NEW: Visual Unittests based on ImageMagick and TortoiseIDiff (for Windows)
- NEW: Pisa raises execptions now if errors occure; with pisaDocument(..., raise_exeception=False) you can turn them off
- UPD: Paragraphs now use the maximum leading to avoid overlapping text
- UPD: Removed "Keep with next" from H1 to H6
- FIX: Sizing of images is now handled better; should better work with PIL
- FIX: Border handling of paragraphs optimized and fixed
- FIX: Images that are higher than the page frame are scaled down to fit
- FIX: Paragraphs only containing   are rendered
- FIX: Problem regarding the order of border style definitions
- FIX: Single
 between two blocks now creates a new line
- FIX: Set table attribute "repeat" to "0"
- FIX: Some <font> attributes did not work as expected
- FIX: Font sizes reworked to behave like browser implmentations
- FIX: Like in most HTML browser table cells now have "valign=middle" and table headers have font weight bold
- FIX: Little fix in CSS parsing
- FIX: Default of <link media=""> was "screen", changed to "all"
- FIX: Command line tools did not install with "easy_install"

Version 3.0.27, 2008-10-04

- INF: License changed from Qt to GPLv2
- INF: Not yet completely combatible with Reportlab 2.2 (errors and borders)
- NEW: Command line tool called "xhtml" ("pisa" still available but will be deprecated with pisa 3.1)
- NEW: EGG for Python 2.6
- NEW: Basic support for Data URI
- NEW: New style -pdf-keep-with-next (does not work with pdf:toc for now)
- UPD: Setup now exclusively works with SetupTools

Version 3.0.26, 2008-08-28

- FIX: Python <2.5 didn't work because of a syntax error

Version 3.0.25, 2008-08-15

- UPD: Made imports more explicit to avoid import recursions
- FIX: <pdf:pagenumber/> didn't work in tables (Roman Lisagor)
- FIX: Images without suffixes have been ignored by pisa (Henning von Bargen)
- INF: Preparations for support of HTML FORM using INPUT, TEXTAREA, SELECT

Version 3.0.24, 2008-07-14

- NEW: Support for separate borders on each side of a paragraph has been added (Robin Dunn)
- NEW: Support for font tag (color, face, size)
- UPD: Handling of margin and padding in paragraphs is improved (Robin Dunn)
- UPD: Updated documentation (CreatePDF, Images)
- FIX: A typo in margin-left has been fixed (Robin Dunn)

Version 3.0.23, 2008-06-26

- UPD: getColor() now understands colors like rgb(255,0,0) (Darryl Dixon)
- FIX: c.warning threw errors if no arguments where passed (Searle)
- FIX: pisa now works with html5lib 0.11.1

Version 3.0.22, 2008-06-06

- UPD: Updated documentation
- UPD: Speed optimizations by removing copy.deepcopy (Darryl Dixon)
- FIX: Small fix in CSS parser

Version 3.0.21, 2008-06-05

- FIX: Used a parameter for html5lib that was not supported by html5lib 0.10
- FIX: Now tested against the latest third party packages: ReportLab 2.1, html5lib 0.10, pyPdf 1.11

Version 3.0.20, 2008-06-02

- NEW: New parameter "encoding" to explicitly set an encoding for the source data
- UPD: Added a programming example to documentation
- FIX: If a Unicode string is passed it will automatically be converted to UTF8
- FIX: Fixes for Google AppEngine support
- FIX: If possible cStringIO will be used instead of StringIO
- FIX: An exception in psaDocument was not handled the right way because a context object was expected

Version 3.0.19, 2008-05-31

- NEW: Support for Google AppEngine
- NEW: Support for page break before and after [not yet tested] (Luka Frelih)
- UPD: Reworked parts of the documentation but not yet completed
- UPD: Optimized the command line tool "pisa"
- FIX: TOC bugs regarding entities and additional tags inside the TOC entry definitions (Luka Frelih)
- FIX: Default logging didn't work with Python<2.5 (Anders J. Munch)
- FIX: StringIO is used instead of cStringIO to avoid encoding problems like the ones we had with GoogleAppEngine

Version 3.0.18, 2008-04-19

- WIN: Updated the windows command line version
- NEW: WSGI support and demo
- NEW: Added simple ASPN Cookbook example

- UPD: Unified setup.py and setup_egg.py (Andreas Gabriel)
- UPD: Better handling of XML and HTML parsing
- UPD: Cleanup of Django sample
- UPD: Cleanup of command line tool options
- UPD: Command line tool doesn't stop batch if error occurred any more
- FIX: 'style' attribute was not evaluated!
- FIX: If a string was passed to pisaDocument it had been converted to StringIO, which was not necessary
- FIX: c.addPara(force=True) works again e.g. for forcing empty pages
- FIX: Better handling of CDATA and Comments
- FIX: Better handling of
- FIX: Removed rsplit() for backward compatibility with Python 2.3
- FIX: Handling of inconsistent HTML anchors
- FIX: TurboGears Demo

Version 3.0.17, 2008-03-23

- NEW: Added CSS support for TOC and updated documentation (Jean Baltus)
- UPD: Added "render_to_pdf" to Django demo (Diego Firmenich)
- UPD: Did some refactoring to make CSS parsing more flexible
- UPD: Removed log.exception for warnings
- FIX: Empty entries in TOC (Jean Baltus)
- FIX: Use correct font for <li> now (reported by Gabor Farkas)

Version 3.0.16, 2008-03-16

- Did some researches about support for languages like Farsi, Arabic and Asian languages. The dir='rtl' feature seems to be quite time intensive to be implemented, maybe I will do it in a later version or on request
- Switched back to HTML parsing by default, but use of XHTML is recommended. Use the option "xhtml" in pisaDocument or "-x" in the command line tool
- Added a decorator for use in Turbogears and CherryPy
- Completely switched to Python logging system
- Created a separate download for the fonts in the "test" directory to reduce the size of the package
- Just use multiBuild if needed e.g. using pdf:toc
- Bugfix: @font-face threw always a warning about font-weight
- Bugfix: List points have to be always in "Helvetica" (Gabor Farkas)
- Bugfix: Obligatory attributes for tag had not been handle the right way
- Bugfix: Marked some old tag based functionalities like pdf:font, pdf:frame and pdf:template as deprecated

Version 3.0.15, 2008-03-13

- Added new package and namespace "ho". With pisa 3.1. we will move away form "sx"
- Added version testing (2.1) for Reportlab Toolkit (Diego Firmenich)

- Added new command <pdf:toc> for support of table of contents, stiling per CSS has not been implemented yet (Jean Baltus)
- Added simple barcode support via command <pdf:barcode> (Diego Firmenich)
- Added Python logging. Name of logger "ho.pisa" and "ho.css". Set debugging level in command line tool by using "-d" for debugging and "-w" for warnings
- Added complete support for CSS "font"
- Modified the version handling and setup system for pisa distributions (had to do with the import errors that where not thrown, reported by Schmitte)
- Updated documentation and added a CSS for HTML version
- Bugfix: CSS "background" URL handling was broken (Luis Bruno)
- Bugfix: CSS "border" now works more standard conform
- Bugfix for compatibility problems with Python 2.3 because of reversed() function
- Bugfix: No exception was thrown if a third party module was missing (Kai Schmitte)
- Bugfix: Changed HTML5 parser from HTMLParser to XHTMLParser so that the custom tags of the "pdf" namespace are handled like expected
- Bugfix: Switched from urllib to urllib2 because status errors (like 404) where not handled (Kees Hink)
- A lot of smaller bugfixes and testings

Version 3.0.14, 2008-02-13

- Added a sample for Unicode support in exotic languages like "farsi" using DejaSans font (Adam Hyde)
- Command line tool generation integrated into setup.py (Andreas Gabriel)
- Bugfix if no path had been set to pisaDocument()
- Bugfix for calculating @frame dimensions
- Bugfix: CSS comments like "//" where allowed (Andreas Gabriel)

Version 3.0.13, 2008-01-22

- Added a demo using cherrypy web server and kid
- Added a demo using django framework
- Modified test-background.html to work with CSS
- Added suport for bold and italic TTF fonts to the @font-face CSS section (Robert Klep)
- Added suport for bold and italic Postscript fonts to the @font-face CSS section
- The @-rules are not need a trailing space after ident any more (Robert Klep)
- Fixed the Windows standalone version to work
- Made the 'sx' folder more sharable by modifying __init__.py
- Changed font-weight so that only values starting with '400'are considered 'bold' (Robert Klep)
- Added "text-indent" style (Robert Klep)
- Added "-pdf-keep-with-next" style to avoid page break between certain elements (Robert Klep)
- Added "-pdf-outline", "-pdf-outline-level" and "-pdf-outline-open" styles to create PDF bookmarks. Per default this is defined for the tags H1 to H6 (Robert Klep)
- New option to overwrite the default CSS definitions of pisa

- New command line options –css
- New command line options –css-dump to get the default CSS definitions. A dump of the recent CSS default may also be found in test/default.css
- Fixed setup.py
- Added EGG installation file support

Version 3.0.12, 2008-01-09

- Moved SVN repository to Holtwick
- Modified copyright notes and links to `http://www.htmltopdf.org`
- Added new table attributes “border”, “bordercolor”, “cellpadding”
- Added support for

Version 3.0.11, 2007-11-13

- New example for loading a page form the web via Python
- New example “test-invoice.html”
- Added support for “align” attribute to <td> and <th>
- Fixed that more than one static frame can use the same named element
- Added -pdf-next-page to specify next page template
- Added -pdf-frame-break: after, before
- Fixed bug for @page without declarations
- Added option for output of errors as PDF (e.g. useful in web applications)
- Set “producer” to “pisa”
- Set author, subject and keywords with <meta>

Version 3.0.10, 2007-11-02

- Fixed some problems with wrong @page and @frame definitions
- New property -pdf-frame-box
- Implemented a pre parser for CSS that cleans up the code with some regular expression, like stripping illegal url `http://...`
- Improved online demo
- First release of binary Windows command line version or pisa
- Fixed some issues with named anchors
- Empty documents are now delivered correctly
- Fixed error on list types
- Fixed problem with debugging infos

Version 3.0.9, 2007-10-31

- Modified setup.py for Chesse Shop
- Added bdist_wininst to setup
- Moved w3c into sx package and added license text
- Modified simple.py demo script

- Clean up for first public release

Version 3.0.8, 2007-10-31

- Added <a name> and a bugfix for ReportLab anchors
- Added <a href>
- More documentation about fonts and new font aliases
- Fixed some bugs in tables
- <hr> now uses ReportLabs implementation
- Margin collapse by using spaceBefore and spaceAfter
- Renamed -pdf-page-size to size (CSS3)

Version 3.0.7, 2007-10-30

- Static frames in @frame
- Wrote layout section in documentation
- Updated the documentation CSS
- Renamed @box to @frame
- Added -pdf-page-size and -pdf-page-orientation
- Added @page and @box
- Fixed some problems with font definitions and Unicode
- Font “Times” does not exist, changed default to “Times-Roman”
- Margins, paddings and borders are only applied in display:block elements

Version 3.0.6, 2007-10-29

- Implemented @font-face
- “font-family” can now handle comma separated font names
- Implemented <pdf:font> for embedding TTF and PS fonts
- <link> looks for rel=”stylesheet”
- Style “white-space” and support for PRE
- Nested lists and ordered lists, Style “list-style-type”
- Prepared parser for @page and @box

Version 3.0.5, 2007-10-25

- Initial implementation of @font-face
- Warnings are only shown if flag -w is set
- Relative @import implementations
- Workaround for styles beginning with asterics like “* font: small”
- Support for color=transparent (threw Exceptions before)
- For @import with now media, is now set media=all
- Fixed the .1 CSS parser problem
- Removed cssutils again because of problems with @import

- Ignore CDATA in style definitions
- New method c.debug and command line option –debug
- Better URL support
- CSS attributes may now start with hyphen for vendor specific styles e.g. “-pdf-page-break”
- Implemented @import
- Implemented @media
- Images are now recalculated to 96DPI too
- 1px = 1/96inch (96dpi) instead of 1px = 1pt = 1/72inch
- Added some new tests like test-css-media.html

Version 3.0.0

- Initial versions of pisa rewrite