pisa 3.0.30

XHTML/HTML/CSS to PDF converter

dirk.holtwick@gmail.com
http://www.xhtml2pdf.com

Table of Contents

Introduction

pisa is a HTML/XHTML/CSS to PDF converter written in Python and based on Reportlab Toolkit, pyPDF, TechGame Networks CSS Library and HTML5lib. The primary focus is not on generating perfect printable webpages but to use HTML and CSS as commonly known tools to generate PDF files within Applications. For example generating documentations (like this one), generating invoices or other office documents etc.

Installation

As pisa is a Python pakage an installed version of Python <http://www.python.org> is needed. For the moment Python 2.3 to 2.5 is supported. For Python 3000 a special version will be needed, because it is not compatible with the 2.x series. A proper version will be made available as soon as Python 3000 becomes stable.

The easiest way to install **pisa** is to use easy_install:

```
$ easy_install pisa
```

But you may also download the source code of **pisa**, then enter the main directory and execute this command (on Linux and MacOS you may prepend a `sudo` command):

```
$ python setup.py install
```

pisa needs also some additional Python packages to be installed to work. Please follow the setup instruction for each package:

- **ReportlabToolkit** 2.1+ (required)
  http://www.reportlab.org/downloads.html
  Provides the Python to PDF conversion functionality
- **html5lib** 0.11.1+ (required)
  http://code.google.com/p/html5lib/
  The parser for HTML and XHTML
- **pyPdf** 1.11+ (optional)
  http://pybrary.net/pyPdf/
  Will be used if you like to place another PDF as a watermark in the background of PDF pages
- **PIL** 1.1.6+ (optional)
  http://www.pythonware.com/products/pil/
  The Python Imaging Library (PIL) is requred by ReportLab for handling of different image formats like GIF and PNG.

Windows precompiled version

For Windows a precompiled version exists that includes Python and all needed libraries. The package contains the file `xhtml2pdf.exe`. Please add the directory where `xhtml2pdf.exe` is placed to the Windows `PATH` variable.

The Windows version is distributed via the Website <http://www.xhtml2pdf.com> in the "Download" section.

Command line

If you do not want to integrate **pisa** in your own application, you may use the command line tool that gives you a simple interface to the features of **pisa**. Just call `xhtml2pdf --help` to get the following help informations:

```
pisa 3.0.30 (Build 2008-12-01)
(C) 2002-2008 Dirk Holtwick <dirk.holtwick@gmail.com>, Germany
Website http://www.htmltopdf.org

USAGE: pisa [options] SRC [DEST]

SRC
 Name of a HTML file or a file pattern using * placeholder.
 If you want to read from stdin use "-" as file name.
 You may also load an URL over HTTP. Take care of putting
 the <src> in quotes if it contains characters like "?".

DEST
 Name of the generated PDF file or "-" if you like
 to send the result to stdout. Take care that the
 destination file is not alread opened by an other
 application like the Adobe Reader. If the destination is
 not writeable a similar name will be calculated automatically.

[options]
 --css, -c:
   Path to default CSS file
 --css-dump:
   Dumps the default CSS definitions to STDOUT
 --debug, -d:
   Show debugging informations
 --encoding:
   the character encoding of SRC. If left empty (default) this
   information will be extracted from the HTML header data
 --help, -h:
   Show this help text
 --quiet, -q:
   Show no messages
 --start-viewer, -s:
   Start PDF default viewer on Windows and MacOSX
   (e.g. AcrobatReader)
 --version:
   Show version information
 --warn, -w:
   Show warnings
 --xml, --xhtml, -x:
   Force parsing in XML Mode
   (automatically used if file ends with ".xml")
 --html:
   Force parsing in HTML Mode (default)
```

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/
```

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. **pisa** uses the HTMLT5lib parser that offers two internal parsing modes: one for HTML and one for XHTML.

When generating the HTML output **pisa** 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 **pisa** like this:

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

The CSS will be dumped into the file `pisa-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
```

Python module

XXX TO BE COMPLETED

The integration into a Python program is quite easy. We will start with a simple "Hello World" example:

```
import ho.pisa as pisa                          (1)

def helloWorld():
 filename = __file__ + ".pdf"                  (2)
 pdf = pisa.CreatePDF(                         (3)
   "Hello <strong>World</strong>",
   file(filename, "wb"))
 if not pdf.err:                               (4)
   pisa.startViewer(filename)                  (5)

if __name__=="__main__":
 pisa.showLogging()                            (6)
 helloWorld()
```

Comments:

(1) Import the **pisa** Python module
(2) Calculate a sample filename. If your demo is saved under `test.py` the filename will be `test.py.pdf`.
(3) The function `CreatePDF` is called with the source and the destination. In this case the source is a string and the destination is a fileobject. Other values will be discussed later (XXX to do!). An object will be returned as result and saved in `pdf`.
(4) The property `pdf.err` is checked to find out if errors occured
(5) If no errors occured a helper function will open a PDF Reader with the resulting file
(6) Errors and warnings are written as log entries by using the Python standard module `logging`. This helper enables printing warnings on the console.

Create PDF

The main function of pisa 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 pisa 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.

Link callback

Images, backgrounds and stylesheets are loaded form an HTML document. Normaly **pisa** 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 reuests.

XXX

Web applications

XXX

Defaults

Some notes on some default values:

- Usually the position (0, 0) in PDF files is found in the lower left corner. For **pisa** it is the upper left corner like it is for HTML.
- The default page size is the German DIN A4 with portrait orientation.
- 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!)

Cascading Style Sheets

pisa supports a lot of Cascading Style Sheet (CSS). The following styles are supported:

```
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
color
display
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
```

And it adds some vendor specific styles:

```
-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
```

Layout Definition

Pages and Frames

Pages can be layouted by using some special CSS at-keywords and properties. All special properties start with `-pdf-` to mark them as vendor specific as defined by CSS 2.1. Layouts may be defined by page using the `@page` keyword. Then text flows in one or more frames which can be defined within the `@page` block by using `@frame`. Example:

```
@page {
 @frame {
   margin: 1cm;
 }
}
```

In the example we define an unnamed page template - though it will be used as the default template - having one frame with `1cm` margin to the page borders. The first frame of the page may also be defined within the `@page` block itself. See the equivalent example:

```
@page {
 margin: 1cm;
}
```

To define more frames just add some more `@frame` blocks. You may use the following properties to define the dimensions of the frame:

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

Here is a more complex example:

```
@page lastPage {
 top: 1cm;
 left: 2cm;
 right: 2cm;
 height: 2cm;
 @frame middle {
   margin: 3cm;
 }
 @frame footer {
```

```
   bottom: 2cm;
   margin-left: 1cm;
   margin-right: 1cm;
   height: 1cm;
 }
}
```

Layout scheme:

```
                top
    +-------------------------+   ---
    |        margin-top       |   /|\
    |    +---------------+    |    |
    |    |               |    |
    |    |               |    |  height
    |    |               |    |
```

By default the Frame uses the whole page and is defined to begin in the upper left corner and end in the lower right corner. Now you can add the position of the frame using `top`, `left`, `bottom` and `right`. If you now add `height` and you have a value other than zero in `top` the `bottom` will be modified. (XXX If you had not defined `top` but `bottom` the `height` will be ...)

Page size and orientation

A page layout may also define the page size and the orientation of the paper using the `size` property as defined in CSS 3. Here is an example defining page size "DIN A5" with "landscape" orientation (default orientation is "portrait"):

```
@page {
 size: a5 landscape;
 margin: 1cm;
}
```

Here is the complete list of valid page size identifiers:

- `a0` ... `a6`
- `b0` ... `b6`
- `letter`
- `legal`
- `elevenseventeen`

PDF watermark/ background

For the use of PDF backgrounds specify the source file in the `background-image` property, like this:

```
@page {
 background-image: url(bg.pdf);
}
```

Static frames

Some frames should be static like headers and footers that means they are on every page but do not change content. The only information that may change is the page number. Here is a simple example that show how to make an element named by ID the content of a static frame. In this case it is the ID `footer`.

```
<html>
<style>
@page {
 margin: 1cm;
 margin-bottom: 2.5cm;
 @frame footer {
   -pdf-frame-content: footerContent;
   bottom: 2cm;
   margin-left: 1cm;
   margin-right: 1cm;
   height: 1cm;
 }
}
</style>
<body>
 Some text
 <div id="footerContent">
   This is a footer on page #<pdf:pagenumber>
 </div>
</body>
</html>
```

For better debugging you may want to add this property for each frame definition: `-pdf-frame-border: 1`. It will paint a border around the frame.

Fonts

By default there is just a certain set of fonts available for PDF. Here is the complete list - and their repective alias names - **pisa** 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**

But you may also embed new font faces 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 font pass just one filename like `<name>.afm` or `<name>.pfb`, the missing one will be calculated automatically.

To define other shapes you may do like this:

```
/* 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;
  font-style: italic;
```

```
}
```

Outlines/ Bookmarks

PDF supports outlines (Adobe calls them "bookmarks"). By default **pisa** 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;
}
```

Table of Contents

It is possible to automatically generate a Table of Contents (TOC) with **pisa**. 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;
}
pdftoc.pdftoclevel1 {
   margin-left: 1em;
}
pdftoc.pdftoclevel2 {
   margin-left: 2em;
   font-style: italic;
}
```

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

Long cells

Pisa 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;
}
```

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.

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 attribute `repeat`. Example:

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

Borders

Borders are supported. Use corresponding CSS styles.

Images

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%; }
```

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.

Barcodes

XXX TO BE WRITTEN

<pdf:barcode>

Custom Tags

pisa provides some custom tags. They are all prefixed by the namespace identifier `pdf:`. As the HTML5 parser used by pisa 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>
```

Tag-Definitions

pdf:barcode

Creates a barcode.

pdf:pagenumber

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

pdf:nexttemplate

Defines the template to be used on the next page.

pdf:nextpage

Create a new page after this position.

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.

pdf:spacer

Creates an object of a specific size.

pdf:toc

Creates a Table of Contents.

License