dompdf Homepage
- digitaljunkies.ca/dompdf

About dompdf
- overview
- features
- limitations

Documentation
- installation
- usage
- api documentation

Examples
- samples
- demo

Support
- FAQ

Send bug reports & support questions to SourceForge or send an email to dompdf at digitaljunkies.ca.

On this page:

- Usage
  - Invoking via the web
  - Invoking via the command line
  - Using the dompdf class directly
  - dompdf class reference
- Inline PHP support

Usage

The dompdf.php script included in the distribution can be used both from the command line or via a web browser. Alternatively, the dompdf class can be used directly.

Invoking dompdf via the web

The dompdf.php script is not intended to be an interactive page. It receives input parameters via $_GET and can stream a PDF directly to the browser. This makes it possible to embed links to the script in a page that look like static PDF links, but are actually dynamically generated. This method is also useful as a redirection target.

dompdf.php accepts the following $_GET variables:

`input_file`	required	a rawurlencoded() path to the HTML file to process. Remote files (http/ftp) are supported if fopen wrappers are enabled.
`paper`	optional	the paper size. Defaults to 'letter' (unless the default has been changed in dompdf_config.inc.php). See include/pdf_adapter.cls.php, or invoke dompdf.php on the command line with the -l switch for accepted paper sizes.
`orientation`	optional	'portrait' or 'landscape'. Defaults to 'portrait'.
`base_path`	optional	the base path to use when resolving relative links (images or CSS files). Defaults to the directory containing the file being accessed. (This option is useful for pointing dompdf at your CSS files even though the HTML file may be elsewhere.)
`output_file`	optional	the rawurlencoded() name of the output file. Defaults to 'dompdf_out.pdf'.
`save_file`	optional	If present (i.e. `isset($_GET["save_file"]) == true');`), output_file is saved locally, Otherwise the file is streamed directly to the client.

One technique for generating dynamic PDFs is to generate dynamic HTML as you normally would, except instead of displaying the output to the browser, you use output buffering and write the output to a temporary file. Once this file is saved, you redirect to the dompdf.php script. If you use a templating engine like Smarty, you can simply do:

```
<?php
$tmpfile = tempnam("/tmp", "dompdf_");
file_put_contents($tmpfile, $smarty->fetch()); // Replace $smarty->fetch()
                                               // with your HTML string

$url = "dompdf.php?input_file=" . rawurlencode($tmpfile) .
       "&paper=letter&output_file=" . rawurlencode("My Fancy PDF.pdf");

header("Location: http://" . $_SERVER["HTTP_HOST"] . "/$url");
?>
```

If you use any stylesheets, you may need to provide the `base_path` option to tell dompdf where to look for them, as they are not likely relative to /tmp ;).

Invoking dompdf via the command line

You can execute dompdf.php using the following command:

```
$ php -f dompdf.php -- [options]
```

(If you find yourself using only the cli interface, you can add `#!/usr/bin/php` as the first line of dompdf.php to invoke dompdf.php directly.)

dompdf.php is invoked as follows:

```
$ ./dompdf.php [options] html_file
```

`html_file` can be a filename, a url if fopen_wrappers are enabled, or the '-' character to read from standard input.

`-h`	Show a brief help message
`-l`	list available paper sizes
`-p size`	paper size; something like 'letter', 'A4', 'legal', etc. Thee default is 'letter'
`-o orientation`	either 'portrait' or 'landscape'. Default is 'portrait'.
`-b path`	the base path to use when resolving relative links (images or CSS files). Default is the directory of html_file.
`-f file`	the output filename. Default is the input `[html_file].pdf`.
`-v`	verbose: display html parsing warnings and file not found errors.
`-d`	very verbose: display oodles of debugging output; every frame in the tree is printed to stdout.

Examples:

```
$ php -f dompdf.php -- my_resume.html
$ ./dompdf.php -b /var/www/ ./web_stuff/index.html
$ echo '<html><body>Hello world!</body></html>' | ./dompdf.php -
```

Using the dompdf class directly

Using the dompdf class directly is fairly straightforward:

```
<?php
require_once("dompdf_config.inc.php");

$html =
  '<html><body>'.
  '<p>Put your html here, or generate it with your favourite '.
  'templating system.</p>'.
  '</body></html>';

$dompdf = new DOMPDF();
$dompdf->load_html($html);
$dompdf->render();
$dompdf->stream("sample.pdf");

?>
```

Below is a summary of the methods available in the dompdf class. For complete details, see the API documentation for the class interface definition.

Method Summary

- *DOMPDF* __construct()
- *string* get_base_path()
- *string* get_host()
- *string* get_protocol()
- *Frame_Tree* get_tree()
- *void* load_html(*string* **$str**)
- *void* load_html_file(*string* **$file**)
- *string* output()
- *void* render()
- *void* set_base_path(*string* **$path**)
- *void* set_host(*string* **$host**)
- *void* set_paper(*string* **$size**, [*string* **$orientation** = "portrait"])
- *void* set_protocol(*string* **$proto**)
- *void* stream(*string $filename*, [*mixed* **$options** = null])

Constructor __construct (line 163)

Class constructor

DOMPDF **__construct**()

get_base_path (line 227)

Returns the base path

string **get_base_path**()

get_canvas (line 234)

Return the underlying Canvas instance (e.g. CPDF_Adapter, GD_Adapter)

Canvas **get_canvas**()

get_host (line 220)

Returns the base hostname

string **get_host**()

get_protocol (line 213)

Returns the protocol in use

string **get_protocol**()

get_tree (line 182)

Returns the underlying Frame_Tree object

Frame_Tree **get_tree**()

load_html (line 272)

Loads an HTML string

Parse errors are stored in the global array _dompdf_warnings.

void **load_html**(*string* **$str**)

- *string* **$str**: HTML text to load

load_html_file (line 245)

Loads an HTML file

Parse errors are stored in the global array _dompdf_warnings.

void **load_html_file**(*string* **$file**)

- *string* **$file**: a filename or url to load

output (line 451)

Returns the PDF as a string

string **output**()

render (line 373)

Renders the HTML to PDF

void **render**()

set_base_path (line 206)

Sets the base path

void **set_base_path**(*string* **$path**)

- *string* **$path**

set_host (line 199)

Sets the base hostname

void **set_host**(*string* **$host**)

- *string* **$host**

set_paper (line 353)

Sets the paper size & orientation

void **set_paper**(*string* **$size**, [*string* **$orientation** = "portrait"])

- *string* **$size**: 'letter', 'legal', 'A4', etc. See CPDF_Adapter::$PAPER_SIZES
- *string* **$orientation**: 'portrait' or 'landscape'

set_protocol (line 192)

Sets the protocol to use (http://, file://, ftp:// etc.)

void **set_protocol**(*string* **$proto**)

- *string* **$proto**

stream (line 441)

Streams the PDF to the client

The file will always open a download dialog. The options parameter controls the output headers. Accepted headers are:

'Accept-Ranges' => 1 or 0 - if this is not set to 1, then this header is not included, off by default. This header seems to have caused some problems despite the fact that it is supposed to solve them, so I am leaving it off by default.

'compress' = > 1 or 0 - apply content stream compression, this is on (1) by default

'Attachment' => 1 or 0 - if 1, force the browser to open a download dialog, on (1) by default

void **stream**(*string* **$filename**, [*array* **$options** = null])

- *string* **$filename**: the name of the streamed file
- *array* **$options**: header options (see above)

Inline PHP Support

dompdf supports two varieties of inline PHP code. All PHP evaluation is controlled by the DOMPDF_ENABLE_PHP configuration option. If it is set to false, then no PHP code is executed. Otherwise, PHP is evaluated in two passes:

The first pass is useful for inserting dynamic data into your PDF. You can do this by embedding <?php ?> tags in your HTML file, as you would in a normal .php file. This code is evaluated prior to parsing the HTML, so you can echo any text or markup and it will appear in the rendered PDF.

The second pass is useful for performing drawing operations on the underlying PDF class directly. You can do this by embedding PHP code within <script type="text/php"> </script> tags. This code is evaluated during the rendering phase and you have access to a few internal objects and operations. In particular, the `$pdf` variable is the current instance of Canvas. Using this object, you can write and draw directly on the current page. Using the `Canvas::open_object()`, `Canvas::close_object()` and `Canvas::add_object()` methods, you can create text and drawing objects that appear on every page of your PDF (useful for headers & footers).

The following variables are defined for you during the second pass of PHP execution:

```
$pdf         the current instance of Canvas
$PAGE_NUM    the current page number
$PAGE_COUNT  the total number of pages in the document
```

For more complete documentation of the Canvas API, see the API documentation.