**Convert HTML to PDF with Dompdf**

[](http://www.sitepoint.com/author/shameerc/)

[Shameer C](http://www.sitepoint.com/author/shameerc/)

Published August 14, 2013

[**Tweet**](https://twitter.com/share?text=Convert+HTML+to+PDF+with+Dompdf&via=sitepointdotcom)[**Subscribe**](https://confirmsubscription.com/h/y/1FD5B523FA48AA2B)

PDF is a standard format originally created by Adobe for representing text and images in a fixed-layout document. It’s not uncommon for a web application to support downloading data, such as invoices or reports, in PDF format, so in this article we’ll go through how we can easily generate PDF documents using PHP.

Dompdf is a great library, capable of generating a PDF from HTML markup and CSS styles (it’s mostly CSS 2.1 compliant and has support for some CSS3 properties). We can decide how our content should look using these familiar technologies, and then easily convert that into a fixed document. The library has many other useful and interesting features, too.

Getting Started

Dompdf is available on GitHub and can be installed using Composer. Getting a Composer-based install up and running correctly is admittedly still a bit tricky, so I recommend just using Git to install Dompdf.

The library requires PHP >= 5.0 with the mbstring and DOM extensions enabled. It also needs some fonts, which are generally available on most machines.

Navigate to wherever you want to put the library and execute:

git clone https://github.com/dompdf/dompdf.git

git submodule init

git submodule update

With Dompdf downloaded, let’s write a short example that will generate a simple PDF.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?php  set\_include\_path(get\_include\_path() . PATH\_SEPARATOR . "/path/to/dompdf");    require\_once "dompdf\_config.inc.php";    $dompdf = new DOMPDF();    $html = <<<'ENDHTML'  <html>   <body>    <h1>Hello Dompdf</h1>   </body>  </html>  ENDHTML;    $dompdf->load\_html($html);  $dompdf->render();    $dompdf->stream("hello.pdf"); |

To use the library in a project, we first pull in dompdf\_config.inc.php which contains most of the Dompdf configuration. It also loads an autoloader and a custom configuration file in which we can override default configuration parameters.

HTML markup is given as a string to the load\_html() method. Alternatively, we can load markup from a file or URL using the load\_html\_file() method. It accepts a filename or the URL of a webpage as its argument.

The render() method renders the HTML into PDF and we are ready to dump the PDF file. stream() sends the resulting PDF as an attachment to the browser. The stream() method has an optional second parameter, an array of options:

* Accept-Ranges – boolean, send “Accept-Ranges” header (false by default).
* Attachment – boolean, send “Content-Disposition: attachment” header forcing the browser to display a save prompt (true by default).
* compress – boolean, enable content compression (true by default).

We’ve generated a very simple PDF here, but that’s not very practical. In reality we’ll often have requirements about paper size, page orientation, character encoding, etc. There are number of configuration options that we can set to make Dompdf more suitable for our real-world needs. All of them are listed and explained indompdf\_config.inc.php which sets them to their default values. You can change these values by updating the the custom configuration file dompdf\_config.custom.inc.php. Some of the important settings are:

* DOMPDF\_DEFAULT\_PAPER\_SIZE – sets the default paper size for the PDF document. Supported paper sizes can be found in include/cpdf\_adapter.cls.php (the default value is “letter”).
* DOMPDF\_TEMP\_DIR – specifies the temporary directory used by Dompdf. Make sure this location is writable by the web server account.
* DOMPDF\_UNICODE\_ENABLED – sets whether the PDF uses Unicode fonts (the default is true).
* DOMPDF\_ENABLE\_REMOTE – enables the inclusion of images or CSS styles from remote sites (default is false).
* DEBUG\_LAYOUT – whether to render a box around each HTML block in the PDF file which is useful for debugging the layout (default is false).

Advanced Usage

Now let’s talk about some advanced uses of Dompdf. Perhaps we want to save the generated PDF document to disk instead of sending it to the browser. Here’s how:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $dompdf = new DOMPDF();  $dompdf->load\_html($html);  $dompdf->render();    $output = $dompdf->output();  file\_put\_contents("/path/to/file.pdf", $output); |

Instead of calling stream() like in the previous example, we use output() which returns the PDF as a string. It too accepts an optional options array, but the only option available is compress (the default is true).

Dompdf also allows us to add a header or footer to the generated PDF by embedding a PHP script in the HTML that it renders. But because processing arbitrary code can pose a security risk if you’re not careful, the configuration value that controls this functionality is off by default. We need to first set the DOMPDF\_ENABLE\_PHP option true.

Once we’ve enabled inline PHP execution, the PDF object will be made available within the script, which we can use to manipulate the page. We can add text, lines, images, rectangles, etc. anywhere inside the page.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | $html = <<<'ENDHTML'  <html>   <body>    <script type="text/php">  if (isset($pdf)) {      // open the PDF object - all drawing commands will      // now go to the object instead of the current page      $footer = $pdf->open\_object();        // get height and width of page      $w = $pdf->get\_width();      $h = $pdf->get\_height();        // get font      $font = Font\_Metrics::get\_font("helvetica", "normal");      $txtHeight = Font\_Metrics::get\_font\_height($font, 8);        // draw a line along the bottom      $y = $h - 2 \* $txtHeight - 24;      $color = array(0, 0, 0);      $pdf->line(16, $y, $w - 16, $y, $color, 1);        // set page number on the left side      $pdf->page\_text(16, $y, "Page: {PAGE\_NUM} of {PAGE\_COUNT}", $font, 8, $color);      // set additional text      $text = "Dompdf is awesome";      $width = Font\_Metrics::get\_text\_width($text, $font, 8);      $pdf->text($w - $width - 16, $y, $text, $font, 8);        // close the object (stop capture)      $pdf->close\_object();        // add the object to every page (can also specify      // "odd" or "even")      $pdf->add\_object($footer, "all");  }    </script>    <h1>Hello Dompdf</h1>   </body>  </html>  ENDHTML; |

The script is embedded directly into the HTML markup, and first opens an object so we can affect the rendering. All drawing will be recorded into that object and we can add this object to all or selected pages (though there are limitations).

Next we fetch the actual width and height of the page to calculate the coordinates of the footer we intend to add. Also, we need to provide a font object while we add text contents. Font\_Metrics::get\_font() lets us to create the object we need. We also take the height of the given font at its font size using get\_font\_height()to calculate positioning for the footer’s contents. The method get\_font\_width() returns the width of our text for the given font and font size which is also used in our calculations.

The line() method draws a line from point (X1,Y1) to (X2,X2). Notice that the color value we provided is not an actual RGB value. The underlying PDF class requires values between 0 and 1 and so we convert the RGB values into these new values. To get a better approximation, you can divide it by 255.

We add the page number for each page using the page\_text() method, which accepts an X and Y position, the text to be added, a font object, font size, and color. Dompdf automatically replaces the values for {PAGE\_NUM} and{PAGE\_COUNT} in each page, and makes $pdf available to us.

When the PDF renders, the footer section will look like this:

dompdf-01

It’s possible to avoid using inline PHP and achieve the same effect directly from PHP, like so:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php  $dompdf = new DOMPDF();  $dompdf->set\_paper("A4");    // load the html content  $dompdf->load\_html($html);  $dompdf->render();  $canvas = $dompdf->get\_canvas();  $font = Font\_Metrics::get\_font("helvetica", "bold");  $canvas->page\_text(16, 800, "Page: {PAGE\_NUM} of {PAGE\_COUNT}", $font, 8, array(0,0,0));  $dompdf->stream("sample.pdf",array("Attachment"=>0)); |

Note that we place the code after calling $dompdf->render() because we are essentially modifying the rendered PDF.

Conclusion

In this article we discussed how to easily convert HTML to PDF using Dompdf. Although Dompdf is a great library, it’s not a bulletproof solution for generating PDF documents; it does has some limitations and issues. Dompdf is not really tolerant of poorly-formed HTML, and large tables can easily cause you to run out of memory. Some basic CSS features like float are not completely supported and there is only limited support for CSS3. If you need features which are not supported by Dompdf, something like wkhtmltopdf might be a better solution for you. Still, Dompdf is fairly simple and suitable for the majority of PDF export needs.

It’s really difficult to explain all of the features provided by a library in article such as this, so be sure to check out the documentation and source code as well to learn about cool features like adding callbacks, using custom fonts, etc. Also, I’m happy to help you within my limited expertise. Feel free to leave your questions and share your experiences in the comments section.

## Tags manual

### Create tags

The tags recognize by Xml2Pdf are define using plugins files. To create a new tag, you must just create a new plugin file and put it in the plugins tags directory who is defined in the config file Xml2Pdf.config.php

eg, to create a tag custom, you must create a file named xml2pdf.tag.custom.php with a class named Xml2PdfTag\_custom.

The plugin class must have the following methods :

* void \_\_construtc(object Pdf, array) or void \_\_construct(object Pdf, array, object Pdf)
* void addContent(string)
* void close()

If the constructor take three parameters, the third will be the parent tag of the curent tag.

example :

1. **class***Xml2PdfTag\_custom*{
2. /\*\*
3. \* parent tag
4. \* @var *object*
5. \*/
6. **public**$parent;
8. /\*\*
9. \* object Pdf.
10. \* @var *object*
11. \*/
12. **public**$pdf;
14. /\*\*
15. \* tag content
16. \* @var *string*
17. \*/
18. **public**$content;
20. /\*\*
21. \* Constructor.
22. \*
23. \* call when the tag custom is open.
24. \*
25. \* @param *object*$pdf Object Pdf
26. \* @param *array*$tagProperties tag properties
27. \* @param *object*$parent tag parent
28. \* @return *void*
29. \*/
30. **public function***\_\_construct*($pdf, $tagProperties, $parent) {
31. // here you can parse the tag properties
32. }
34. /\*\*
35. \* add the content
36. \*
37. \* @param *string*$content tag content
38. \* @return *void*
39. \*/
40. **public function***addContent*($content) {
41. // here you can manage the content
42. }
44. /\*\*
45. \* call when the tag is closed
46. \*
47. \* @return *void*
48. \*/
49. **public function***close*() {
50. // here you can maange the actions to do when the tag is closed
51. }
52. }

To the following XML :

<parent foo="foovalue">

<custom bar="barvalue">custom tag content.</custom>

</parent>

The PHP code will be :

1. // <parent foo="foovalue">
2. **require\_once**('xml2pdf.tag\_parent.php');
3. $parentTag = **new***Xml2PdfTag\_parent*($pdfObject, **array**('FOO'=>'foovalue'));
5. // <custom bar="barvalue">
6. **require\_once**('xml2pdf.tag\_custom.php');
7. $customTag = **new***Xml2PdfTag\_custom*($pdfObject, **array**('BAR'=>'barvalue'), $parentTag);
9. // custom tag content.
10. $customTag->[addContent](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/Xml2PdfTag.html#methodaddContent)('custom tag content.');
12. // </custom>
13. $customTag->[close](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/Xml2PdfTextTag.html#methodclose)();
15. // </parent>
16. $parentTag->[close](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/Xml2PdfTextTag.html#methodclose)();

### Existing tags

The existing tags plugins are :

* [Tag pdf](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.pdf.pkg.html)
* [Tag body](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.body.pkg.html)
* [Tag page](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.page.pkg.html)
* [Tag header](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.header.pkg.html)
* [Tag footer](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.footer.pkg.html)
* [Tag text](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.text.pkg.html)
* [Tag paragraph](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.paragraph.pkg.html)
* [Tag date](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.date.pkg.html)
* [Tag numerotation](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.numerotation.pkg.html)
* [Tag image](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.image.pkg.html)
* [Tag barcode](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.barcode.pkg.html)
* [Tag link](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.link.pkg.html)
* [Tag ln](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.ln.pkg.html)
* [Tag stylesheet](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.stylesheet.pkg.html)
* [Tag literal](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.literal.pkg.html)
* [Tag table, tt, th, tr, td](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.table.pkg.html)
* [Tag graph](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.graph.pkg.html)
* [Tag value](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.value.pkg.html)
* [Tag line](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.line.pkg.html)
* [Tag point](http://xml-2-pdf.sourceforge.net/phpdoc/html/Xml2Pdf/tutorial_Xml2Pdf.Tag.point.pkg.html)