GitHub This repository Search Explore Features Enterprise Blog Sign up Sign in

bpampuch / pdfmake

Watch 48 Star 416 Fork 92

Client/server side PDF printing in pure JavaScript http://pdfmake.org

329 commits | 2 branches | 24 releases | 24 contributors

branch: master • pdfmake / +

Don't waste performance when there is no page break before function ...

jthoenes authored 10 days ago latest commit b31c381126

build	webpack + gulp	18 days ago
dev-playground	Stripped down playground for pdfmake development	2 months ago
examples	image wrapping: add integration test for images	2 months ago
libs	Update libs/fileSaver.js	8 months ago
src	Don't waste performance when there is no page break before function	10 days ago
tests	Update to pdfkit 0.7.1 - it will set Helvetica as F1 on construction.	17 days ago
.editorconfig	Adds an editorconfig	3 months ago
.gitignore	webpack + gulp	18 days ago
.jshintrc	js hint	a year ago
.npmignore	browser extensions	a year ago
.travis.yml	browserify bundle options fix	2 months ago
Gruntfile.js	Merge pull request #221 from TW-QEN/qen-fixToManyCharacterIssue	2 months ago
LICENSE	Initial commit	a year ago
README.md	fixes readme typo for italic style setting	2 months ago
bower.json	updated bower ignore array	18 days ago
gulpfile.js	webpack + gulp	18 days ago
package.json	Update to pdfkit 0.7.1 - it will set Helvetica as F1 on construction.	17 days ago
webpack.config.js	webpack w-i-p	18 days ago

Code
Issues 73
Pull requests 2
Wiki
Pulse
Graphs
HTTPS clone URL
https://github.com/l
You can clone with HTTPS or Subversion.
Clone in Desktop
Download ZIP

README.md

pdfmake

build passing | npm package 0.1.18 | bower package ?.?.?

Client/server side PDF printing in pure JavaScript

Check out the playground

Features

- line-wrapping,
- text-alignments (left, right, centered, justified),
- numbered and bulleted lists,
- tables and columns
  - auto/fixed/star-sized widths,
  - col-spans and row-spans,
  - headers automatically repeated in case of a page-break,
- images and vector graphics,
- convenient styling and style inheritance,

- page headers and footers:
  - static or dynamic content,
  - access to current page number and page count,
- background-layer
- page dimensions and orientations,
- margins,
- custom page breaks,
- font embedding,
- support for complex, multi-level (nested) structures,
- helper methods for opening/printing/downloading the generated PDF.

Getting Started

This document will walk you through the basics of pdfmake and will show you how to create PDF files in the browser. If you're interested in server-side printing check the examples folder.

To begin with the default configuration, you should include two files:

- **pdfmake.min.js**,
- **vfs_fonts.js** - default font definition (it contains Roboto, you can however use custom fonts instead)

```
<!doctype html>
<html lang='en'>
<head>
  <meta charset='utf-8'>
  <title>my first pdfmake example</title>
  <script src='build/pdfmake.min.js'></script>
  <script src='build/vfs_fonts.js'></script>
</head>
<body>
...
```

You can get both files using bower:

```
bower install pdfmake
```

or copy them directly from the build directory from the repository.

Document-definition-object

pdfmake follows a declarative approach. It basically means, you'll never have to calculate positions manually or use commands like: `writeText(text, x, y)` , `moveDown` etc..., as you would with a lot of other libraries.

The most fundamental concept to be mastered is the document-definition-object which can be as simple as:

```
var docDefinition = { content: 'This is an sample PDF printed with pdfMake' };
```

or become pretty complex (having multi-level tables, images, lists, paragraphs, margins, styles etc...).

As soon as you have the document-definition-object, you're ready to create and open/print/download the PDF:

```
// open the PDF in a new window
pdfMake.createPdf(docDefinition).open();

// print the PDF (not working in this version, will be added back in a couple of days)
// pdfMake.createPdf(docDefinition).print();
```

```
// download the PDF
pdfMake.createPdf(docDefinition).download();
```

Styling

pdfmake makes it possible to style any paragraph or its part:

```
var docDefinition = {
  content: [
    // if you don't need styles, you can use a simple string to define a paragraph
    'This is a standard paragraph, using default style',

    // using a { text: '...' } object lets you set styling properties
    { text: 'This paragraph will have a bigger font', fontSize: 15 },

    // if you set the value of text to an array instead of a string, you'll be able
    // to style any part individually
    {
      text: [
        'This paragraph is defined as an array of elements to make it possible to ',
        { text: 'restyle part of it and make it bigger ', fontSize: 15 },
        'than the rest.'
      ]
    }
  ]
};
```

Style dictionaries

It's also possible to define a dictionary of reusable styles:

```
var docDefinition = {
  content: [
    { text: 'This is a header', style: 'header' },
    'No styling here, this is a standard paragraph',
    { text: 'Another text', style: 'anotherStyle' },
    { text: 'Multiple styles applied', style: [ 'header', 'anotherStyle' ] }
  ],

  styles: {
    header: {
      fontSize: 22,
      bold: true
    },
    anotherStyle: {
      italics: true,
      alignment: 'right'
    }
  }
};
```

To have a deeper understanding of styling in pdfmake, style inheritance and local-style-overrides check STYLES1, STYLES2 and STYLES3 examples in playground.

Columns

By default paragraphs are rendered as a vertical stack of elements (one below another). It is possible however to divide available space into columns.

```
var docDefinition = {
  content: [
    'This paragraph fills full width, as there are no columns. Next paragraph however consists
    {
      columns: [
        {
```

```
            // auto-sized columns have their widths based on their content
            width: 'auto',
            text: 'First column'
          },
          {
            // star-sized columns fill the remaining space
            // if there's more than one star-column, available width is divided equally
            width: '*',
            text: 'Second column'
          },
          {
            // fixed width
            width: 100,
            text: 'Third column'
          },
          {
            // % width
            width: '20%',
            text: 'Fourth column'
          }
        ],
        // optional space between columns
        columnGap: 10
      },
      'This paragraph goes below all columns and has full width'
    ]
};
```

Column content is not limited to a simple text. It can actually contain any valid pdfmake element. Make sure to look at the COLUMNS example in playground.

Tables

Conceptually tables are similar to columns. They can however have headers, borders and cells spanning over multiple columns/rows.

```
var docDefinition = {
  content: [
    {
      table: {
        // headers are automatically repeated if the table spans over multiple pages
        // you can declare how many rows should be treated as headers
        headerRows: 1,
        widths: [ '*', 'auto', 100, '*' ],

        body: [
          [ 'First', 'Second', 'Third', 'The last one' ],
          [ 'Value 1', 'Value 2', 'Value 3', 'Value 4' ],
          [ { text: 'Bold value', bold: true }, 'Val 2', 'Val 3', 'Val 4' ]
        ]
      }
    }
  ]
};
```

All concepts related to tables are covered by TABLES example in playground.

Lists

pdfMake supports both numbered and bulleted lists:

```
var docDefinition = {
  content: [
    'Bulleted list example:',
    {
      // to treat a paragraph as a bulleted list, set an array of items under the ul key
      ul: [
```

```
        'Item 1',
        'Item 2',
        'Item 3',
        { text: 'Item 4', bold: true },
      ]
    },

    'Numbered list example:',
    {
      // for numbered lists set the ol key
      ol: [
        'Item 1',
        'Item 2',
        'Item 3'
      ]
    }
  ]
};
```

Headers and footers

Page headers and footers in pdfmake can be: *static* or *dynamic*.

They use the same syntax:

```
var docDefinition = {
  header: 'simple text',

  footer: {
    columns: [
      'Left part',
      { text: 'Right part', alignment: 'right' }
    ]
  },

  content: (...)
};
```

For dynamically generated content (including page numbers and page count) you can pass a function to the header or footer:

```
var docDefinition = {
  footer: function(currentPage, pageCount) { return currentPage.toString() + ' of ' + pageCount
  header: function(currentPage, pageCount) {
    // you can apply any logic and return any valid pdfmake element

    return { text: 'simple text', alignment: (currentPage % 2) ? 'left' : 'right' };
  },
  (...)
};
```

Background-layer

The background-layer will be added on every page.

```
var docDefinition = {
  background: 'simple text',

  content: (...)
};
```

It may contain any other object as well (images, tables, ...) or be dynamically generated:

```
var docDefinition = {
  background: function(currentPage) {
```

```
    return 'simple text on page ' + currentPage
  },

  content: (...)
};
```

Margins

Any element in pdfMake can have a margin:

```
(...)
// margin: [left, top, right, bottom]
{ text: 'sample', margin: [ 5, 2, 10, 20 ] },

// margin: [horizontal, vertical]
{ text: 'another text', margin: [5, 2] },

// margin: equalLeftTopRightBottom
{ text: 'last one', margin: 5 }
(...)
```

Stack of paragraphs

You could have figured out by now (from the examples), that if you set the `content` key to an array, the document becomes a stack of paragraphs.

You'll quite often reuse this structure in a nested element, like in the following example:

```
var docDefinition = {
  content: [
    'paragraph 1',
    'paragraph 2',
    {
      columns: [
        'first column is a simple text',
        [
          // second column consists of paragraphs
          'paragraph A',
          'paragraph B',
          'these paragraphs will be rendered one below another inside the column'
        ]
      ]
    }
  ]
};
```

The problem with an array is that you cannot add styling properties to it (to change fontSize for example).

The good news is - array is just a shortcut in pdfMake for { stack: [] }, so if you want to restyle the whole stack, you can do it using the expanded definition:

```
var docDefinition = {
  content: [
    'paragraph 1',
    'paragraph 2',
    {
      columns: [
        'first column is a simple text',
        {
          stack: [
            // second column consists of paragraphs
            'paragraph A',
            'paragraph B',
            'these paragraphs will be rendered one below another inside the column'
          ],
```

```
          fontSize: 15
        }
      ]
    }
  ]
};
```

Images

This is simple. Just use the `{ image: '...' }` node type.

JPEG and PNG formats are supported.

```
var docDefinition = {
  content: [
    {
      // you'll most often use dataURI images on the browser side
      // if no width/height/fit is provided, the original size will be used
      image: '...encodedContent...'
    },
    {
      // if you specify width, image will scale proportionally
      image: '...encodedContent...',
      width: 150
    },
    {
      // if you specify both width and height - image will be stretched
      image: '...encodedContent...',
      width: 150,
      height: 150
    },
    {
      // you can also fit the image inside a rectangle
      image: '...encodedContent...',
      fit: [100, 100]
    },
    {
      // if you reuse the same image in multiple nodes,
      // you should put it to to images dictionary and reference it by name
      image: 'mySuperImage'
    },
    {
      // under NodeJS (or in case you use virtual file system provided by pdfmake)
      // you can also pass file names here
      image: 'myImageDictionary/image1.jpg'
    }
  ],

  images: {
    mySuperImage: '...content...'
  }
};
```

Page dimensions, orientation and margins

```
var docDefinition = {
  // a string or { width: number, height: number }
  pageSize: 'A5',

  // by default we use portrait, you can change it to landscape if you wish
  pageOrientation: 'landscape',

  // [left, top, right, bottom] or [horizontal, vertical] or just a number for equal margins
  pageMargins: [ 40, 60, 40, 60 ],
};
```

If you set `pageSize` to a string, you can use one of the following values:

- '4A0', '2A0', 'A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10',
- 'B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9', 'B10',
- 'C0', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9', 'C10',
- 'RA0', 'RA1', 'RA2', 'RA3', 'RA4',
- 'SRA0', 'SRA1', 'SRA2', 'SRA3', 'SRA4',
- 'EXECUTIVE', 'FOLIO', 'LEGAL', 'LETTER', 'TABLOID'

To change page orientation within a document, add a page break with the new page orientation.

```
{
  pageOrientation: 'portrait',
  content: [
    {text: 'Text on Portrait'},
    {text: 'Text on Landscape', pageOrientation: 'landscape', pageBreak: 'before'},
    {text: 'Text on Landscape 2', pageOrientation: 'portrait', pageBreak: 'after'},
    {text: 'Text on Portrait 2'},
  ]
}
```

Coming soon

Hmmm... let me know what you need ;)

The goal is quite simple - make pdfmake useful for a looooooooot of people and help building responsive HTML5 apps with printing support.

There's one thing on the roadmap for v2 (no deadline however) - make the library hackable, so you can write plugins to:

- extend document-definition-model (with things like { chart: ... }),
- add syntax translators (like the provided [...] -> { stack: [...] }
- build custom DSLs on top of document-definition-model (this is actually possible at the moment).

License

MIT

pdfmake is based on a truly amazing library pdfkit.org - credits to @devongovett

big thanks to @yelouafi for making this library even better

© 2015 GitHub, Inc. Terms Privacy Security Contact

Status API Training Shop Blog About