page download as pdf javascript

jsPDF.

You can catch me on twitter: @MrRio or head over to my company's website for consultancy.

Creating your first document.

See examples/basic.html. There's a live editor example at index.html.

Head over to jsPDF.com for details or here for our most recent live editor and examples.

Checking out the source.

Credits.

Big thanks to Daniel Dotsenko from Willow Systems Corporation for making huge contributions to the codebase.

Everyone else that's contributed patches or bug reports. You rock.

License.

Page download as pdf javascript.

A library to generate PDFs in JavaScript.

You can catch me on twitter: @MrRio or head over to my company's website for consultancy.

Recommended: get jsPDF from npm:

Alternatively, load it from a CDN:

Or always get latest version via unpkg.

The dist folder of this package contains different kinds of files:

jspdf.es.*.js : Modern ES2015 module format. jspdf.node.*.js : For running in Node. Uses file operations for loading/saving files instead of browser APIs. jspdf.umd.*.js : UMD module format. For AMD or script-tag loading. polyfills*.js : Required polyfills for older browsers like Internet Explorer. The es variant simply imports all required polyfills from core-js , the umd variant is self-contained.

Usually it is not necessary to specify the exact file in the import statement. Build tools or Node automatically figure out the right file, so importing "jspdf" is enough.

Then you're ready to start making your document:

If you want to change the paper size, orientation, or units, you can do:

Running in Node.js.

Other Module Formats.

Some functions of jsPDF require optional dependencies. E.g. the html method, which depends on html2canvas and, when supplied with a string HTML document, dompurify . JsPDF loads them dynamically when required (using the respective module format, e.g. dynamic imports). Build tools like Webpack will automatically create separate chunks for each of the optional dependencies. If your application does not use any of the optional dependencies, you can prevent Webpack from generating the chunks by defining them as external dependencies:

In Vue CLI projects, externals can be defined via the configureWebpack or chainWebpack properties of the vue.config.js file (needs to be created, first, in fresh projects).

In Angular projects, externals can be defined using custom webpack builders.

In React (create-react-app) projects, externals can be defined by either using react-app-rewired or ejecting.

jsPDF can be imported just like any other 3rd party library. This works with all major toolkits and frameworks. jsPDF also offers a typings file for TypeScript projects.

You can add jsPDF to your meteor-project as follows:

jsPDF requires modern browser APIs in order to function. To use jsPDF in older browsers like Internet Explorer, polyfills are required. You can load all required polyfills as follows:

Alternatively, you can load the prebundled polyfill file. This is not recommended, since you might end up loading polyfills multiple times. Might still be nifty for small applications or quick POCs.

Use of Unicode Characters / UTF-8:

The 14 standard fonts in PDF are limited to the ASCII-codepage. If you want to use UTF-8 you have to integrate a custom font, which provides the needed glyphs. jsPDF supports .ttf-files. So if you want to have for example Chinese text in your pdf, your font has to have the necessary Chinese glyphs. So, check if your font supports the wanted glyphs or else it will show garbled characters instead of the right text.

To add the font to jsPDF use our fontconverter in /fontconverter/fontconverter.html. The fontconverter will create a js-file with the content of the provided ttf-file as base64 encoded string and additional code for jsPDF. You just have to add this generated js-File to your project. You are then ready to go to use setFont-method in your code and write your UTF-8 encoded text.

Alternatively you can just load the content of the *.ttf file as a binary string using fetch or XMLHttpRequest and add the font to the PDF file:

Since the merge with the yWorks fork there are a lot of new features. However, some of them are API breaking, which is why there is an API-switch between two API modes:

In "compat" API mode, jsPDF has the same API as MrRio's original version, which means full compatibility with plugins. However, some advanced features like transformation matrices and patterns won't work. This is the default mode. In "advanced" API mode, jsPDF has the API you're used from the yWorks-fork version. This means the availability of all advanced features like patterns, FormObjects, and transformation matrices.

You can switch between the two modes by calling.

JsPDF will automatically switch back to the original API mode after the callback has run.

Please check if your question is already handled at Stackoverflow https://stackoverflow.com/questions/tagged/jspdf. Feel free to ask a question there with the tag jspdf .

Feature requests, bug reports, etc. are very welcome as issues. Note that bug reports should follow these guidelines:

A bug should be reported as an mcve Make sure code is properly indented and formatted (Use ``` around code blocks) Provide a runnable example. Try to make sure and show in your issue that the issue is actually related to jspdf and not your framework of choice.

jsPDF cannot live without help from the community! If you think a feature is missing or you found a bug, please consider if you can spare one or two hours and prepare a pull request. If you're simply interested in this project and want to help, have a look at the open issues, especially those labeled with "bug".

You can find information about building and testing jsPDF in the contribution guide.

Big thanks to Daniel Dotsenko from Willow Systems Corporation for making huge contributions to the codebase. Thanks to Ajaxian.com for featuring us back in 2009. Our special thanks to GH Lee (sphilee) for programming the ttf-file-support and providing a large and long sought after feature Everyone else that's contributed patches or bug reports. You rock.

Inserting pages into a PDF with Acrobat JavaScript.

Learn how to automate the task of inserting pages from one PDF into another using Acrobat JavaScript.

By Thom Parker – February 12, 2009.

Scope: Acrobat 5.0 and later Category : Automation Skill Level: Intermediate and Advanced Prerequisites: Basic Acrobat JavaScript Programming.

One of the most common document-preparation tasks is inserting pages from one PDF into another PDF. Whether it's simply appending documents or inserting individual pages into special locations, the Acrobat user interface can be very cumbersome when this is a frequent task. Fortunately, page insertion is also one of the easiest tasks to automate and has been part of the Acrobat JavaScript model since version 5.

About page insertion.

Page insertion is performed with the doc.insertPages() function. This function takes four input arguments: the page number where insertion starts, a path to the PDF that is the source of the insertion pages, and the start and ending pages to insert from the source PDF. To start, we need two PDF files for an insertion example, a source document and a destination document. You can create your own, or download the following two files to a single folder on your hard drive.

If you create your own sample files, make sure each has four easily identifiable pages. Also, make sure both files are in the same folder.

Open " InsertExampleDest.pdf" in Acrobat Professional. The JavaScript insertion function always acts on the currently open PDF.

Inserting a page into a PDF is a major document modification, so it is not an operation that can be done in Adobe Reader, and it requires a Privileged Context. At its simplest, a Privileged Context means this operation cannot be done from a Document Level script, i.e., documents can't insert pages into themselves. This has to be done from an Application Level script.

For this example we'll be running our code from the JavaScript Console window. The JavaScript Console gives our code privilege, so it's handy for running cut-and-paste automation code like the insertion function, but it is also the essential tool for Acrobat JavaScript development and should be used for all code development. If the console is not enabled, or you have not used it before, then please read the JavaScript Console article.

After opening InsertExampleDest.pdf, display the JavaScript Console from the Advanced > Document Processing menu item or by pressing Ctrl + j. Enter the following code into the console window:

At a minimum, only the path argument, cPath , has to be specified; and since both files are in the same folder, we only need to specify the name of the source file, not the full path. Notice the object notation being used to pass the arguments into the function. This is an Acrobat DOM feature, not a core JavaScript feature. It only works on functions that are part of the Acrobat JavaScript Model. It's useful because it saves us from having to specify the other optional arguments.

Before running this code, open up the Pages panel (Figure 1). Run the code by placing the cursor on the same line as the code in the JavaScript Console and pressing Ctrl+Enter . The source PDF has a blue-ish background so it is easy to see that all pages from the source PDF are inserted after the first page (Figure 2). Because the page number of the insertion point was not included, the insertPages() function uses the default value, which is the first page in the document. Page numbers used in Acrobat JavaScript always start at 0, so the first page in the PDF is page number 0.

Figure 1 –Pages before insertion.

Figure 2 –Pages after insertion.

To insert pages into a specific location, the page after which pages are to be inserted is specified. This is the nPage argument. Before running any of the following code from the JavaScript Console, first revert the open PDF to its unchanged state by selecting Revert from the File menu. Also, note that a few of the following examples have multiple lines. To run these from the JavaScript Console, select all the lines before pressing Ctrl + Enter.

To append the entire source PDF to the end of the currently open PDF, use this code:

To insert the entire source PDF in front of the pages in the open PDF, use the code below. Notice that "-1" is used as the insertion point.

To insert the entire source PDF after page two, use the following code. Notice the code does not use the object notation. The first argument is the insertion point and the second argument is the source file path, so the object notation is unnecessary.

To insert a single page from the source file, we need another argument, the starting page in the source PDF. In the code below, page three from the source PDF is inserted after page two in the open PDF.

To insert a range of pages from the source file, we need yet another argument, the ending page number. In this example, pages three and four of the source PDF are inserted after page one of the currently open PDF.

Using the example scripts.

As stated earlier, the insertPages() function does not work in Adobe Reader and requires a Privileged Context. It, and other functions like it--such as replacePages() , deletePages() , movePage() , and addWaterMarkFromFile() — are all used to automate document-preparation tasks. As such, they are not intended for use in document scripts.

In the examples above, we ran the code by copying and pasting the scripts from this article into the JavaScript Console window. In fact, for doing simple automation tasks, it's a good idea to place all your favorite scripts into a text document from which you can copy and paste.

To insert pages into a group of files, you would use a Batch Sequence. Batch Sequences are a Privileged Context, so any of the example code could be copied directly into a Batch Sequence.

A more interesting and useful way to run an automation script is with an Acrobat Toolbar Button or Menu Item. However, using one of these options requires that the code be enclosed in a trusted function. Code for creating toolbar buttons and trusted functions can be found in this article, Applying PDF security with Acrobat JavaScript.

For more information on functions used in this article, see the Acrobat JavaScript Reference and the Acrobat JavaScript Guide.

Click on the Documentation tab and scroll down to the JavaScript section.

How To Download HTML Content as PDF File Using JavaScript.

How To Download HTML Content as PDF File Using JavaScript | In this article I'm going to explain you one the most interesting and useful tutorial, that is download the HTML content as in the PDF format onclick using jsPDF jquery file.

Using jsPDF library, you can download the div contents as a PDF file. This is a HTML5 client-side solution for generating PDFs.

Simply include library in your <head>, generate your PDF using the many built-in functions, then create a button to trigger the download.

HTML Code.

JavaScript Code.

Final Code.

We hope this article helped you to easily learn how to download html content as pdf file using javascript. You may also want to see – How To Change The Default WordPress Email Address.

If you liked this article, then please share to social networking site. You can also find us on Twitter,Facebook and Instagram.

Editorial Staff.

Editorial Staff at Veewom is a Team of Experts led by Bharat Makwana, We also manage other popular sites like HostingMirror, Onroid and NaukriRadar.

Print.js.

A tiny javascript library to help printing from the web.

PDF Printing.

Print.js was primarily written to help us print PDF files directly within our apps, without leaving the interface, and no use of embeds. For unique situations where there is no need for users to open or download the PDF files, and instead, they just need to print them.

One scenario where this is useful, for example, is when users request to print reports that are generated on the server side. These reports are sent back as PDF files. There is no need to open these files before printing them. Print.js offers a quick way to print these files within our apps.

Example.

Add a button to print a PDF file located on your hosting server:

For large files, you can show a message to the user when loading files.

Print Large PDF (5mb file) Print Extra Large PDF (16mb file)

The library supports base64 PDF printing:

Print base64 PDF.

HTML Printing.

Sometimes we just want to print selected parts of a HTML page, and that can be tricky. With Print.js, we can easily pass the id of the element that we want to print. The element can be of any tag, as long it has a unique id. The library will try to print it very close to how it looks on screen, and at the same time, it will create a printer friendly format for it.

Example.

Add a print button to a HTML form:

Print.js accepts an object with arguments. Let's print the form again, but now we will add a header to the page:

Print Form with Header.

Image Printing.

Print.js can be used to quickly print any image on your page, by passing the image url. This can be useful when you have multiple images on the screen, using a low resolution version of the images. When users try to print the selected image, you can pass the high resolution url to Print.js.

Example.

Load images on your page with just the necessary resolution you need on screen: