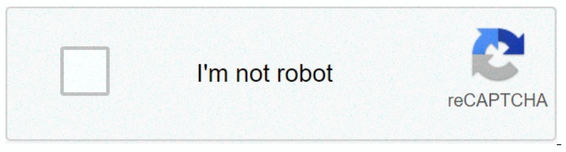
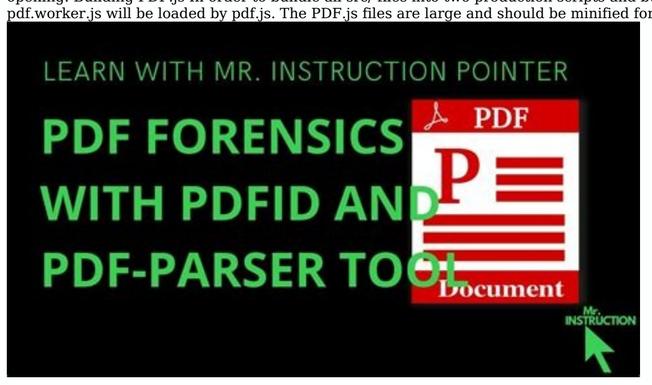


Continue

Pdf parser in javascript

Pdf js examples. Pdf javascript examples.

The pdf-parser is a parser of PDF content and layout information with pdf.js. Getting the Code To get a local copy of the current code, clone it using git: $ git clone $ cd pdf-parser Next, install Node.js via the official package or via nvm. install all dependencies for pdf-parser: $ npm install Example The example is in the file example/node/pdf2json.js. If you are in the root directory, you can use node example/node/pdf2json to run the example. Or you can cd example/node and then node pdf2json. Usage var pdfParser = require('pdf-parser'); var PDF_PATH = 'test.pdf'; pdfParser.pdf2json(PDF_PATH, function (error, pdf) { if(error != null){ console.log(error); }else{ console.log(JSON.stringify(pdf)); } }); The tool can convert pdf to json as bellow: { "pages":[{ "width":612, "height":792, "pageId":0, "texts":[{ "text":"Hello World", "direction":"ltr", //from left to right "width":52.81644000000001, "height":27.96, "top":278.69, "left":296.81, "transform":[27.96,0,0,27.96,296.81,278.69], "fontSize":27.96, "fontName":"Times", "fontOriginName":"TimesNewRomanPSMT", "bold":false, "italic":false, "black":false, "color":"[68,113,196]" }] }] } Java Usage The usage of how to call the pdf-parser in java : java-pdf-parser Questions File an issue: PDF.js is a Portable Document Format (PDF) viewer that is built with HTML5. PDF.js is community-driven and supported by Mozilla. Our goal is to create a general-purpose, web standards-based platform for parsing and rendering PDFs. Contributing PDF.js is an open source project and always looking for more contributors. To get involved, visit: Feel free to stop by our Matrix room for questions or guidance. Getting Started Online demo Please note that the "Modern browsers" version assumes native support for the latest JavaScript features; please also see this wiki page. Browser Extensions Firefox PDF.js is built into version 19+ of Firefox. Chrome Build Your Own - Get the code as explained below and issue gulp chromium. Then open Chrome, go to Tools > Extension and load the (unpackaged) extension from the directory build/chromium. Getting the Code To get a local copy of the current code, clone it using git: $ git clone $ cd pdf.js Next, install Node.js via the official package or via nvm. You need to install the gulp package globally (see also gulp's getting started): $ npm install -g gulp-cli If everything worked out, install all dependencies for PDF.js: Finally, you need to start a local web server as some browsers do not allow opening PDF files using a file:// URL. Run: and then you can open: Please keep in mind that this requires a modern and fully up-to-date browser; refer to Building PDF.js for non-development usage of the PDF.js library. It is also possible to view all test PDF files on the right side by opening: Building PDF.js In order to bundle all src/ files into two production scripts and build the generic viewer, run: If you need to support older browsers, run: This will generate pdf.js and pdf.worker.js in the build/generic/build/ directory (respectively build/generic-legacy/build/). Both scripts are needed but only pdf.js needs to be included since pdf.worker.js will be loaded by pdf.js. The PDF.js files are large and should be minified for production. Using PDF.js in a web application To use PDF.js in a web application you can choose to use a pre-built version of the library or to build it from source. We supply pre-built versions for usage with NPM and Bower under the pdfjs-dist name.



For more information and examples please refer to the wiki page on this subject. Including via a CDN PDF.js is hosted on several free CDNs: Learning You can play with the PDF.js API directly from your browser using the live demos below: More examples can be found in the examples folder. Some of them are using the pdfjs-dist package, which can be built and installed in this repo directory via gulp dist-install command. For an introduction to the PDF.js code, check out the presentation by our contributor Julian Viereck: More learning resources can be found at: The API documentation can be found at: Questions Check out our FAQs and get answers to common questions: Talk to us on Matrix: mozilla.org File an issue: Follow us on Twitter: @pdfjs I provide 2 days of Hacking PDF training at HITB Amsterdam. This is one of the methods I teach. Maarten Van Horenbeeck posted a diary entry (July 2008) explaining how scripts and data are stored in PDF documents (using streams), and demonstrated a Perl script to decompress streams.

A couple of months before, I had started developing my pdf-parser tool, and Maarten’s diary entry motivated me to continue adding features to pdf-parser. Extracting and decompressing a stream (for example containing a JavaScript script) is easy with pdf-parser. You select the object that contains the stream (example object 5: -o 5) and you “filter” the content of the stream (-f). The command is: pdf-parser.py –o 5 –f sample.pdf In PDF jargon, streams are compressed using filters. You have all kinds of filters, for example ZLIB DEFLATE, but also lossy compressions like JPEG. pdf-parser supports a couple of filters, but not all, because the implementation of some of them (mostly the lossy ones) differs between vendors and PDF applications. A recent article published by Virus Bulletin on JavaScript stored inside a lossy stream gave me the opportunity to implement a method I had worked out manually. The problem: you need to decompress a stream and you have no decompression algorithm. The solution: you use the PDF application to decompress the stream. The method: you create a new PDF document with the stream as embedded file, and then save the embedded file using the PDF application. The detailed method: when you need to decompress a stream for which you have no decompressor (or no decompressor identical to the target application), you create a new PDF document into which you include the object with the stream as an embedded file. PDF documents support embedded files. For example, if you have a PDF document explaining a financial method, you can include a spreadsheet in the PDF document as an embedded file. The embedded file is stored as an object with a stream, and the compression can be any method supported by the PDF application. Crafting this PDF document with embedded file manually requires many manipulations and calculations, and is thus a very good candidate for automation. Figure: this PDF embeds a file called vbanner2.jpg With pdf-parser, you can use this method as follows: Create a Python program that generates the PDF document with embedded file. Use pdf-parser like this (in this example, the data stream you want to decompress is in object 5 of PDF file sample.pdf): pdf-parser.py –generateembedded 5 sample.pdf > embedded.py Execute the Python program to create the PDF file: embedded.py embedded.pdf Open the created PDF file embedded.pdf with the target application (Adobe Reader for the Virus Bulletin example), and save the embedded file to disk The saved file contains the decompressed stream You can find my PDF tools here. Remark: the generated Python program requires my module mPDF.py, which can also be found on my PDF tools page. Remark 2: don’t use this method when the stream contains an exploit for the decompressor.