PDF.js

A general-purpose, web standards-based platform for parsing and rendering PDFs.

Download (getting_started/#download) Demo (web/viewer.html)

Github Project (https://github.com/mozilla/pdf.js)

Getting Started

An introduction to PDF.js with examples.

Introduction

Before downloading PDF.js please take a moment to understand the different layers of the PDF.js project.

Layer	About
Core	The core layer is where a binary PDF is parsed and interpreted. This layer is the foundation for all subsequent layers. It is not documented here because using it directly is considered an advanced usage and the API is likely to change. For an example of using the core layer see the PDF Object Browser (https://github.com/brendandahl/pdf.js.utils/tree/master/browser)
Display	The display layer takes the core layer and exposes an easier to use API to render PDFs and get other information out of a document. This API is what the version number is based on.
Viewer	The viewer is built on the display layer and is the UI for PDF viewer in Firefox and the other browser extensions within the project. It can be a good starting point for building your own viewer. *However, we do ask if you plan to embed the viewer in your own site, that it not just be an unmodified version. Please re-skin it or build upon it.*

Download

Pre-built

Includes the generic build of PDF.js and the viewer.

Stable (v1.0.712) (https://github.com/mozilla/pdf.js/releases/download/v1.0.712/pdfjs-1.0.712-dist.zip)

Beta (v1.0.907) (https://github.com/mozilla/pdf.js/releases/download/v1.0.907/pdfjs-1.0.907-dist.zip)

Source

To get a local copy of the current code, clone it using git:

```
$ git clone git://github.com/mozilla/pdf.js.git pdfjs
$ cd pdfjs
```

File Layout Overview

Prebuilt

```
├── LICENSE
├── build/
│   ├── pdf.js                              - display layer
│   └── pdf.worker.js                       - core layer
└── web/
    ├── cmaps/                              - character maps(required by core)
    ├── compatibility.js                    - polyfills for missing features
    ├── compressed.tracemonkey-pldi-09.pdf - test pdf
    ├── debugger.js                         - helpful pdf debugging features
    ├── images/                             - images for the viewer and annotation icon
s
    ├── l10n.js                             - localization
    ├── locale/                             - translation files
    ├── viewer.css                          - viewer style sheet
    ├── viewer.html                         - viewer html
    └── viewer.js                           - viewer layer
```

Source

```
├── AUTHORS
├── CONTRIBUTING.md
├── LICENSE
├── README.md
├── build/                                  - output of build steps (empty)
├── docs/                                   - this
├── examples/                               - bare bones examples
├── extensions/                             - various browser extensions
├── external/                               - third party code
├── l10n/                                   - translation files
├── make.js                                 - build script
├── package.json
├── src/
│   ├── core/                               - core layer
│   ├── display/                            - display layer
│   ├── images/
│   ├── pdf.js                              - wrapper file that everything is bundled i
nto
│   ├── shared/                             - shared code between core and display laye
rs
│   └── worker_loader.js                    - used for developer builds to load worker
files
├── test/                                   - reference, unit, and font tests
└── web/                                    - viewer layer
```

Trying the Viewer

With the prebuilt or source version open `web/viewer.html` in a browser and the test pdf should load. Note: the worker is not enabled for file:// urls, so use a server. If you're using the source build and have node, you can run `node make server`.

More Information

For a further walkthrough of a minimal viewer see the hello world example. More documentation can be found in our wiki (https://github.com/mozilla/pdf.js/wiki) too.

Frequently Asked Questions

yurydelendik edited this page on Sep 23 · 64 revisions

Can I specify a different PDF in the default viewer?

You can modify the `DEFAULT_URL` variable in the `web/viewer.js` file or you can append the `?file=` query string to the viewer URL, e.g. `http://mozilla.github.com/pdf.js/web/viewer.html?file=compressed.tracemonkey-pldi-09.pdf`.

Can I load a PDF from another server (cross domain request)?

Not by default, but it is possible. PDF.js runs with the same permissions as any other JavaScript code, which means it cannot do cross origin requests (see Same origin policy and an example). There are some possible ways to get around this such as using CORS (see also unsafe headers issue and Access-Control-Expose-Headers issue) or setting up a proxy on your server that will feed PDF.js the PDF file. Both workarounds are out of the scope of the PDF.js project and we will not provide code to do either.

What browsers are supported?

The goal is to support all HTML5 compliant browsers, but since feature support varies per browser/version our support for all PDF features varies as well. If you want to support more browsers than Firefox you'll need to include compatibility.js which has polyfills for missing features. Find the list of features needed for PDF.js to properly work and browser tests for those features at Required Browser Features. In general, the support is below:

Browser	Supported	Automated Testing	Notes
Firefox Stable	yes	Windows and Linux	
Chrome Stable	yes	Windows and Linux	
Opera Stable	yes	none	
Android	limited	none	Android's Web Browser version 4.0 or below lacks a number of features or has defects, e.g. in typed arrays or HTTP range requests
Safari	limited	none	Safari (desktop and mobile) lacks a number of features or has defects, e.g. in typed arrays or HTTP range requests
IE9	limited	none	IE9 lacks a number of features and most notably typed arrays which causes subpar performance.
<=IE8	NO	none	IE8 and below are missing too many features to be supported.

What browsers have extensions (and where can I find install procedures)?

There is currently a Firefox, Chromium and Opera extension. The Firefox extension is well supported and actively worked on. The Chromium extension is maintained by a PDF.js contributor. The Opera extension can be found here. For installing the Firefox or Chromium extension, please refer to the readme.

I know JavaScript and want to contribute to the project. How do I start?

First, you need to prepare your fork and setup the development environment. Don't forget to read the Contributing page. Second, make yourself familiar with the PDF format and PDF.js internals. Third, if you don't already have a certain issue you want to fix, choose one from the open issues labeled 5-good-beginner-bug. Last, submit a pull request for the review. *During any part of the process we recommend to communicate with the PDF.js team on #pdfjs IRC channel at irc.mozilla.org if you have questions or need to find a reviewer.*

Is it possible to add annotations to a PDF?

PDF.js is mainly written for *reading* PDF files, not editing them. Because of that we don't yet support adding any kind of annotations. We do however support rendering a number of annotation types for viewing.

What are the PDF.js keyboard shortcuts?

(warning, the following list may be incomplete)

Navigation

- **next page:** `n`, `j`, right arrow key, click in presentation mode
- **previous page:** `p`, `k`, left arrow key, `Shift` + click in presentation mode

The `home`, `end`, `page up`, `page down` and all arrow keys can be used to navigate the document.

Viewer controls

User interface buttons or `ctrl` + mouse wheel can be used to change the zooming level, but keyboard shortcuts are also available:

- **zoom in:** `ctrl` + `+`, `ctrl` + `=`
- **zoom out:** `ctrl` + `-`
- **restore normal zoom:** `ctrl` + `0`
- **rotate the document clockwise:** `r`
- **rotate counterclockwise:** `shift` + `r`
- **presentation mode:** `ctrl` + `alt` + `p` (does not work in IE11)
- **toggle hand tool:** `h`
- **move focus to the 'go to page' box:** `ctrl` + `alt` + `g`

(replace ctrl with meta on some configurations)

The PDF.js files are too big. Can you provide minified versions of JS files?

The only supported minifier as of now is Google Closure Compiler (see https://developers.google.com/closure/compiler). You can build a minified version of PDF.js using the following command:

```
CLOSURE_COMPILER="../path/to/closure/compiler.jar" node make minified
```

It is known that other minifiers might break PDF.js code if advanced options are used (see #710 or #2479). It's safe to use minifiers in whitespace/comments removal mode.

Is there a pre-built version PDF.js available?

Yes. Please see http://mozilla.github.io/pdf.js/getting_started/ page for details. Also the code for the website at http://mozilla.github.io/pdf.js is located in the "gh-pages" branch. You can clone it using `git clone -b gh-pages https://github.com/mozilla/pdf.js.git pdfjs-gh-pages` or download the archive.

There are also generic PDF.js library builds available at https://github.com/mozilla/pdfjs-dist. These builds can be installed via npm `npm install pdfjs-dist` or bower `bower install pdfjs-dist`.

PDF.js does not render my files correctly. Can I report an issue?

Yes. The issues are used to track both bugs filed by users and specific work items for developers. Try to file one issue per problem observed.

Please specify valid title (e.g. "Glyph spacing is incorrect" instead of "PDF.js does not work") and provide more details about the issue: link to the PDF, location in the PDF, screenshot, browser version, operating system, PDF.js version and JavaScript console warning/error messages. The issues that do not have enough details provided will be closed as invalid/incomplete.

I know that my PDFs are corrupted. Will PDF.js attempt to display it?

Yes. PDF.js will attempt to recover usable PDF data (pages, content or fonts) and display the document. Please report the issue (see above) and we will take a look.

I have a really great idea. Where is the best place to record it?

The best place is our dev-pdf-js@lists.mozilla.org mailing list. You can subscribe to it using lists.mozilla.org or Google Groups. This way you will reach not only developers. As an alternative, you can join our weekly engineering meeting to discuss new ideas for the project.

The issue tracking system is designed to record a single technical problem. A bug report is something where a developer/contributor can work on. The GitHub issue tracker is not a good place for general, not well thought out or unworkable ideas. Most likely a discussion-type issue will not be addressed for a long time or closed as invalid.

I'm developing a custom solution based on PDF.js core library. Can you help me?

We are glad to hear that and will try to help you, but first check examples at https://github.com/mozilla/pdf.js#learning and search existing issues. If this does not help, please prepare short well-documented example that demonstrate the problem and make it accessible online on your website, jsbin, etc. before opening a new issue or contacting us on the IRC channel -- keep in mind that just code snippets won't help us troubleshoot the problem. The issues that do not provide enough details will be closed as invalid/incomplete (see reporting issue above).

Please periodically check or subscribe to our dev-pdf-js@lists.mozilla.org mailing list to be informed about changes in the PDF.js architecture/design or security announcements.

What is a latest stable version of PDF.js?

PDF.js is a general-purpose library to parse and render PDFs. At the moment it's included in the number of projects such as Firefox, Firefox OS, Chromium Extension, etc. We are recording our changes to the library with Github pull requests. Also the log of the changes is available from the git log.

The version number consists of three digits: the major release number, minor release number and build number. The major and minor number are selected when some major milestone is reached. The build number is incremented by one each time when new a

commit is pushed to the master branch. For sanity check, we accompany each version number with the SHA number of the latest commit.

We are moving fast and trying to land as much good stuff as we can review and test. The generic viewer and development of version of Firefox PDF Viewer extension always contain the latest PDF.js build and available for testing.

During cooldown period, about once or twice in 6 weeks, we push our library to the Firefox Nightly channel. We decided to tag/mark our master branch each time we do that, and at this point a beta release is created. To promote a latest beta to a stable release, we listen for feedback (via github, bugzilla, mailing list, or IRC) from the users and projects that use PDF.js library. If no critical issues (e.g. a build is unusable, majority of the documents cannot be rendered, etc.) appeared, we promote the build as stable. Otherwise we either discard the release by replacing it by new beta or redo the build with commits that will fix a critical issue.

What types of PDF files are slow in PDF.js? Can I optimize a PDF file to make PDF.js faster?

Typically, PDFs with a smaller file size will be rendered faster and it depends on how big a single page is. The amount of pages does not affect the performance. It's essential that you optimize your documents for the web. See Optimize a PDF from Adobe's website for more information. There are more improvement techniques that we can suggest:

1. Avoid using high resolution images – 150 dpi resolution for scanned images shall be enough for screens, especially for low powered devices;
2. Try to use JPEG encoding for color images/photos in RGB colorspace when possible;
3. Avoid using expensive compositions/effects such as transitions/masking – flatten transparency;
4. Avoid using PDF generators (or don't create content) that produce ineffective PDF output (e.g. LibreOffice creates a lots of tiny images for vector elements/pictures it does not understand);
5. If there is such a setting, use web-optimized PDF output / linearization;
6. Fix or don't produce corrupted PDFs that do not conform to the PDF32000 specification.