Asciidoctor PDF | Asciidoctor

Authors

Asciidoctor PDF is a native PDF converter for AsciiDoc. It bypasses the requirement to generate an interim format such as DocBook, Apache FO, or LaTeX. Instead, you can use it to convert directly from AsciiDoc to PDF. Its aim is to take the pain out of creating PDF documents from AsciiDoc.

Highlights

- Direct AsciiDoc to PDF conversion
- Configuration-driven style and layout
- SVG support
- PDF document outline (i.e., bookmarks)
- Table of contents page(s)
- Document metadata (title, authors, subject, keywords, etc)
- Internal cross reference links
- Syntax highlighting with Rouge, Pygments, or CodeRay
- Page numbering
- Customizable running content (header and footer)
- “Keep together” blocks (i.e., page breaks avoided in certain block content)
- Orphaned section titles avoided
- Autofit verbatim blocks (as permitted by base_font_size_min setting)
- Table border settings honored
- Font-based icons
- Custom (TTF) fonts
- Double-sided printing mode (margins alternate on recto and verso pages)

Prerequisites

All that’s needed is Ruby (1.9.3 or above, though 2.3.x or above is recommended) and a few Ruby gems, which we explain how to install in the next section.

To check if you have Ruby available, use the `ruby` command to query the version installed:

By default, Asciidoctor PDF automatically installs the latest version of Prawn if you don’t already have Prawn installed. This will fail on older versions of Ruby. To workaround, you need to install certain dependencies first.

Starting with Prawn 2.0.0, Prawn requires Ruby >= 2.0.0 during installation. Therefore, to use Asciidoctor PDF with Ruby 1.9.3, you must first explicitly install the following dependencies to lock the versions:

```
$ gem install prawn --version 1.3.0
```

```
  gem install addressable --version 2.4.0
  gem install prawn-svg --version 0.21.0
  gem install prawn-templates --version 0.0.3
```

You can then proceed with installation of Asciidoctor PDF on Ruby 1.9.3.

Starting with Prawn 2.2.0, Prawn requires Ruby >= 2.1.0 during installation. Therefore, to use Asciidoctor PDF with Ruby 2.0.0, you must first explicitly install the following dependencies to lock the versions:

```
$ gem install prawn --version 2.1.0
  gem install prawn-svg --version 0.26.0
  gem install prawn-templates --version 0.0.4
```

For all other versions of Ruby, you can simply install the Asciidoctor PDF gem. It will transitively install the required dependencies.

System Encoding

Asciidoctor assumes you're using UTF-8 encoding. To minimize encoding problems, make sure the default encoding of your system is set to UTF-8.

If you're using a non-English Windows environment, the default encoding of your system may not be UTF-8. As a result, you may get an `Encoding::UndefinedConversionError` or other encoding issues when invoking Asciidoctor. To solve these problems, we recommend at least changing the active code page in your console to UTF-8.

Once you make this change, all your Unicode headaches will be behind you.

Getting Started

Install the Published Gem

Asciidoctor PDF is published as a pre-release on RubyGems.org. First, make sure you have satisfied the prerequisites. Then, you can install the published gem using the following command:

```
$ gem install asciidoctor-pdf --pre
```

If you want to syntax highlight source listings, you'll also want to install Rouge, Pygments, or CodeRay. Choose one (or more) of the following:

Rouge (preferred, minimum version: 1.11.1)

Pygments

```
$ gem install pygments.rb
```

You then activate syntax highlighting for a given document by adding the `source-highlighter` attribute to the document header (Rouge shown):

```
:source-highlighter: rouge
```

Run the Application

Assuming all the required gems install properly, verify you can run the `asciidoctor-pdf` script:

If you see the version of Asciidoctor PDF printed, you're ready to use Asciidoctor PDF!

Let's grab an AsciiDoc document to distill and start putting Asciidoctor PDF to use.

An Example AsciiDoc Document

If you don't already have an AsciiDoc document, you can use the basic-example.adoc file found in the examples directory of this project.

basic-example.adoc

```
= Document Title
Doc Writer <doc@example.com>
:doctype: book
:reproducible:
//:source-highlighter: coderay
:source-highlighter: rouge
:listing-caption: Listing
// Uncomment next line to set page size (default is A4)
//:pdf-page-size: Letter

A simple http://asciidoc.org[AsciiDoc] document.

== Introduction

A paragraph followed by a simple list with square bullets.

[square]
* item 1
* item 2

Here's how you say "`Hello, World!`" in Prawn:

.Create a basic PDF document using Prawn
[source,ruby]
----
require 'prawn'

Prawn::Document.generate 'example.pdf' do
  text 'Hello, World!'
end
----
```

It’s time to convert the AsciiDoc document directly to PDF.

Convert AsciiDoc to PDF

> You’ll need the `rouge` gem installed to run this example since it uses the `source-highlighter` attribute with the value of `rouge`.

Converting to PDF is a simple as running the `asciidoctor-pdf` script using Ruby and passing our AsciiDoc document as the first argument.

```
$ asciidoctor-pdf basic-example.adoc
```

This command is just a shorthand way of running:

```
$ asciidoctor -r asciidoctor-pdf -b pdf basic-example.adoc
```

The `asciidoctor-pdf` command just saves you from having to remember all those flags. That’s why we created it.

When the script completes, you should see the file *basic-example.pdf* in the same directory. Open the *basic-example.pdf* file with a PDF viewer to see the result.

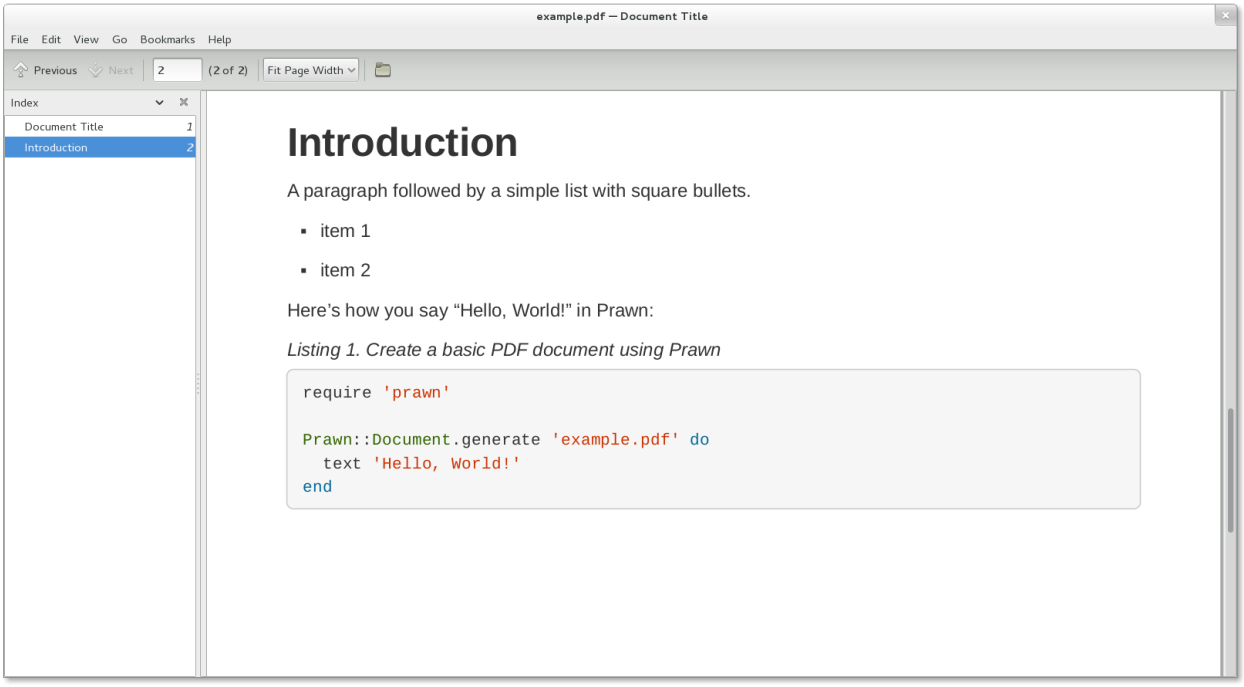

Figure 1. Example PDF document rendered in a PDF viewer

The pain of the DocBook toolchain should be melting away about now.

Themes

The layout and styling of the PDF is driven by a YAML configuration file. To learn how the theming system works and how to create and apply custom themes, refer to the Asciidoctor PDF Theme Guide. You can use the built-in theme files, which you can find in the *data/themes* directory, as examples.

Support for Non-Latin Languages

Asciidoctor can process the full range of characters in the UTF-8 character set. That means you can write your document in any language, save the file with UTF-8 encoding, and expect Asciidoctor to convert the text properly. However, you may notice that certain characters for certain languages, such as Chinese, are missing in the PDF. Read on to find out why and how to address it.

If you're writing in a non-Latin language, you need to use a dedicated theme that provides the necessary fonts. For example, to produce a PDF from a document written in a CJK language such as Chinese, you need to use a CJK theme. You can get such a theme by installing the `asciidoctor-pdf-cjk-kai_gen_gothic` gem. See the asciidoctor-pdf-cjk-kai_gen_gothic project for detailed instructions.

Using a dedicated theme is necessary because PDF is a "bring your own font" kind of system. In other words, the font you provide must provide glyphs for all the characters. There's no one font that supports all the worlds languages (though some, like Noto Serif, certainly come close). Even if there were such a font, bundling that font with the main gem would make the package enormous. It would also severely limit the style choices in the default theme, which targets Latin-based languages.

Therefore, we're taking the strategy of creating separate dedicated theme gems that target each language family, such as CJK. The base theme for CJK languages is provided by the asciidoctor-pdf-cjk project and a concrete implementation provided by the asciidoctor-pdf-cjk-kai_gen_gothic project that's based on the kai_gen_gothic font. Of course, you're free to follow this model and create your own theme gem that uses fonts of your choice.

Font-Based Icons

You can use icons in your PDF document using any of the following icon sets:

You can enable font-based icons by setting the following attribute in the header of your document:

If you want to override the font set globally, also set the `icon-set` attribute:

```
:icons: font
:icon-set: pf
```

Here's an example that shows how to use the Amazon icon from the payment font (pf) icon set in a sentence:

```
Available now at icon:amazon[].
```

You can use the `set` attribute on the icon macro to override the icon set for a given icon.

```
Available now at icon:amazon[set=pf].
```

In addition to the sizes supported in the HTML backend (lg, 1x, 2x, etc), you can enter any relative value in the size attribute (e.g., 1.5em, 150%, etc).

You can enable use of fonts during PDF generation (instead of in the document header) by passing the `icons` attribute to the `asciidoctor-pdf` command.

```
$ asciidoctor-pdf -a icons=font -a icon-set=octicon sample.adoc
```

Icon-based fonts are handled by the `prawn-icon` gem. To find a complete list of available icons, consult the prawn-icon repository.

Image Paths

Images are resolved relative to the value of the `imagesdir` attribute at the time the converter is run. This is effectively the same as how the built-in HTML converter works when the `data-uri` attribute is set. The `imagesdir` is blank by default, which means images are resolved relative to the input document.

If the image is an SVG, and the SVG includes a nested raster image (PNG or JPG) with a relative path, that path is resolved relative to the directory that contains the SVG.

The converter will refuse to include an image if the target is a URI unless the `allow-uri-read` attribute is enabled via the CLI or API.

Image Scaling

Since PDF is a fixed-width canvas, you almost always need to specify a width to get the image to fit properly on the page. There are five ways to specify the width of an image, listed here in order of precedence:

Attribute Name	Description
pdfwidth	The display width of the image as an absolute size (e.g., 2in), percentage of the content area width (e.g., 75%), or percentage of the page width (e.g., 100vw). If a unit of measurement is not specified (or not recognized), pt (points) is assumed. *Intended to be used for the PDF converter only.*
scaledwidth	The display width of the image as an absolute size (e.g., 2in) or percentage of the content area width (e.g., 75%). If a unit of measurement is not specified, % (percentage) is assumed. If a unit of measurement is recognized, pt (points) is assumed. *Intended to be used for print output such as PDF.*
image_width key from theme	Accepts the same values as pdfwidth. *Only applies to block images.*
width	The unitless display width of the image (assumed to be pixels), typically matching the intrinsic width of the image. If the width exceeds the content area width, the image is scaled down to the content area width.

Attribute Name	Description
unspecified	If you don't specify one of the aforementioned width settings, the intrinsic width of the image is used (the px value is multiplied by 75% to convert to pt, assuming canvas is 96 dpi) unless the width exceeds the content area width, in which case the image is scaled down to the content area width.

The image is always sized according to the explicit or intrinsic width and its height is scaled proportionally. The height of the image is ignored by the PDF converter unless the height of the image exceeds the content height of the page. In this case, the image is scaled down to fit on a single page.

If you want a block image to align to the boundaries of the page (not the content margin), specify the `align-to-page` option (e.g., `opts="align-to-page"`). This is most useful when using vw units because you can make the image cover the entire width of the page.

Images in running content also support the `fit` attribute. If the value of this attribute is `contain`, the image size is increased or decreased to fill the available space while preserving its aspect ratio. If the value of this attribute is `scaled-down`, the image size is first resolved in the normal way, then scaled down further to fit within the available space, if necessary.

Page background images are automatically scaled to fit the bounds of the page. Any explicit sizing is ignored for those images.

Using the pdfwidth Attribute

The pdfwidth attribute is the recommended way to set the image size for the PDF output. This attribute is provided for two reasons. First, the fixed-width canvas often calls for a width that is distinct from other output formats, such as HTML. Second, this attribute allows the width to be expressed using a variety of units.

The pdfwidth attribute supports the following units:

- pt (default)
- in
- cm
- mm
- px
- pc
- vw (percentage of page width)
- % (percentage of content area width)

In all cases, the width is converted to pt.

Specifying a default width

If you want to scale all block images that don't have a pdfwidth or scaledwidth attribute specified, assign a value to the image_width key in your theme.

Inline Image Sizing

Inline images can be sized in much the same way as block images (using the pdfwidth, scaledwidth or width attributes), with the following exceptions:

- The viewport width unit (i.e., vw) is not recognized in this context.
- The image will be scaled down, if necessary, to fit the width and height of the content area.

- Inline images do not currently support a default width controlled from the theme.

If the resolved height of the image is less than or equal to 1.5 times the line height, the image won't disrupt the line height and is centered vertically in the line. This is done to maximize the use of available space. Once the resolved height exceeds this amount, the height of the line is increased (by increasing the font size of the invisible placeholder text) to accommodate the image. In this case, the surrounding text will be aligned to the bottom of the image. If the image height exceeds the height of the page, the image will be scaled down to fit on a single page (this may cause the image to advance to the subsequent page).

Fonts in SVG Images

Asciidoctor PDF uses prawn-svg to embed SVGs in the PDF document, including SVGs generated by Asciidoctor Diagram.

Actually, it's not accurate to say that prawn-svg embeds the SVG. Rather, prawn-svg is an SVG *renderer*. prawn-svg translates an SVG into native PDF text and graphic objects. You can think of the SVG as a sequence of drawing commands. The result becomes indistinguishable from other PDF objects.

What that means for text is that any font family used for text in the SVG *must* be registered in the Asciidoctor PDF theme file (and thus with Prawn). Otherwise, Prawn will fallback to using the closest matching built-in (afm) font from PDF (e.g., sans-serif becomes Helvetica). Recall that afm fonts only support basic Latin. As we like to say, PDF is bring your own font.

If you're using Asciidoctor Diagram to generate SVGs to embed in the PDF, you likely need to specify the default font the diagramming tool uses. Let's assume you are making a plantuml diagram.

To set the font used in the diagram, first create a file named *plantuml.cfg* and populate it with the following content:

```
skinparam defaultFontName Noto Serif
```

You can choose any font name that is registered in your Asciidoctor PDF theme file. When using the default theme, your options are "Noto Serif", "M+ 1mn", and "M+ 1p Fallback".

Next, pass that path to the `plantumlconfig` attribute in your AsciiDoc document (or set the attribute via the CLI or API):

```
:plantumlconfig: plantuml.cfg
```

Clear the cache of your diagrams and run Asciidoctor PDF with Asciidoctor Diagram enabled. The diagrams will be generated using Noto Serif as the default font, and Asciidoctor PDF will know what to do.

The bottom line is this: If you're using fonts in your SVG, and you want those fonts to be preserved, those fonts must be defined in the Asciidoctor PDF theme file.

Supporting Additional Image File Formats

In order to embed an image into a PDF, Asciidoctor PDF must understand how to read it. To perform this work, Asciidoctor delegates to the underlying libraries. Prawn provides support for reading JPG and PNG images. prawn-svg brings support for SVG images. Without any additional libraries, those are the only supported image file formats.

If you need support for additional image formats, such as GIF, TIFF, or interlaced PNG—and you don't want to convert those images to a supported format—you must install the prawn-gmagick Ruby gem. prawn-gmagick is an extension for Prawn based on GraphicsMagick that adds support for all the image formats recognized by that library. prawn-gmagick has the added benefit of significantly reducing the time it takes to generate a PDF containing a lot of images.

The prawn-gmagick gem uses native extensions to compile against GraphicsMagick. This system prerequisite limits installation to Linux and OSX. Please refer to the README for prawn-gmagick to learn how to install it.

Once this gem is installed, Asciidoctor automatically switches over to it to handle embedding of all images. In addition to support for more additional image file formats, this gem also speeds up image processing considerably,

so we highly recommend using it if you can.

STEM Support

Unlike the built-in HTML converter, Asciidoctor PDF does not provide native support for STEM blocks and inline macros. That's because Asciidoctor core doesn't actually process STEM equations. In the HTML output, Asciidoctor relies on the JavaScript-based MathJax library to parse and render the equations in the browser. All the HTML converter does is wrap the equations so MathJax is able to find them.

In order to insert a rendered equation into the PDF, the toolchain has to parse the equations and convert them to a format the PDF writer (Prawn) can understand. That typically means converting to an image.

The recommended solution is an extension named Asciidoctor Mathematical, which we'll cover here. Another solution still under development uses Mathoid to convert STEM equations to images. Mathoid is a library that invokes MathJax using a headless browser. That prototype can be found in the Asciidoctor extensions lab.

Asciidoctor Mathematical

Asciidoctor Mathematical is an extension for Asciidoctor that provides alternate processing of STEM blocks and inline macros. After the document has been parsed, the extension locates all the STEM blocks and inline macros, converts the equations to images using Mathematical, then replaces them with image nodes. Conversion then proceeds as normal.

Asciidoctor Mathematical is a Ruby gem that uses native extensions. It has a few system prerequisites which limit installation to Linux and OSX. Please refer to the installation section in the Asciidoctor Mathematical README to learn how to install it.

Once Asciidoctor Mathematical is installed, you just need to enable it when invoking Asciidoctor PDF using the `-r` flag:

```
$ asciidoctor-pdf -r asciidoctor-mathematical sample.adoc
```

If you're invoking Asciidoctor via the API, all you need to do is require the gem before invoking Asciidoctor:

```
require 'asciidoctor-mathematical'

Asciidoctor.convert_file 'sample.adoc', safe: :safe
```

To get the best quality equations, and the maximize speed of conversion, we recommend configuring Asciidoctor Mathematical to convert equations to SVG. You control this setting using the `mathematical-format` AsciiDoc attribute:

```
$ asciidoctor-pdf -r asciidoctor-mathematical -a mathematical-format=svg sample.adoc
```

Refer to the README for Asciidoctor Mathematical to learn about additional settings and options.

Shaded Blocks and Performance

Certain blocks are rendered with a shaded background, such as verbatim (listing, literal, and source), sidebar, and example blocks. In order to calculate the dimensions of the shaded region, Asciidoctor PDF has to "dry run" the block to determine how many pages it consumes. While this strategy has a low impact when processing shorter blocks, it can drastically deteriorate performance when processing a block that spans dozens of pages. As a general rule of thumb, you should avoid using shaded blocks which span more than a handful of pages.

Autofitting Text

Verbatim blocks often have long lines that don't fit within the fixed width of the PDF canvas. And unlike on the web, the PDF reader cannot scroll horizontally to reveal the overflow text. Therefore, the long lines are forced to wrap. Wrapped lines can make the verbatim blocks hard to read or even cause confusion.

To help address this problem, Asciidoctor PDF provides the `autofit` option on all verbatim (i.e., literal, listing and

source) blocks to attempt to fit the text within the available width. When the `autofit` option is enabled, Asciidoctor PDF will decrease the font size until the longest line fits without wrapping.

The font size will not be decreased beyond the value of the `base_font_size_min` key specified in the PDF theme. If that threshold is reached, lines may still wrap.

Here's an example of the autofit option enabled on a source block:

```
[source%autofit,java]
----
@SessionScoped
public class WidgetRepository {
    @GET
    @Produces("application/json")
    public List<String> listAll(@QueryParam("start") Integer start, @QueryParam("max") Integer max) {
        ...
    }
}
----
```

If you want to enable the autofit option globally, set the `autofit-option` document attribute in the document header (or somewhere before the relevant blocks):

Printing Page Ranges

The print dialog doesn't understand the page numbers labels (which appear in the running content). Instead, it only considers physical pages. Therefore, to print a range of pages as they are labeled in the document, you need to add the number of front matter pages (i.e., the non-numbered pages) to the page number range in the print dialog.

For example, if you only want to print the first 5 pages labeled with a page number (e.g., 1-5), and there are 2 pages before the page labeled as page 1, you need to add 2 to both numbers in the range, giving you a physical page range of 3-7. That's the range you need to enter into the print dialog.

Index Catalog

Asciidoctor PDF supports generating an index catalog that itemizes all index terms defined in the document, allowing the reader to navigate the document by keyword.

To get Asciidoctor PDF to generate an index, simply add an level-1 section flagged with the `index` style near the end of your document. The converter will automatically populate the catalog with the list of index terms in the document, organized by first letter.

You can use any text you want for the title of the section. The only restriction is that no index terms may be defined below this section.

Although the catalog is generated automatically, you have to mark the index terms manually. However, you could use an extension, such as a TreeProcessor, to automatically mark index terms.

Optional Scripts

Asciidoctor PDF also provides a shell script that invokes GhostScript (`gs`) to optimize and compress the generated PDF with minimal impact on quality. You must have Ghostscript installed to use it.

Here's an example usage:

```
$ ./bin/optimize-pdf basic-example.pdf
```

The command will generate the file *example-optimized.pdf* in the current directory.

The `optimize-pdf` script currently requires a Bash shell (Linux, OSX, etc). We plan to rewrite the script in Ruby so it works across platforms (see issue #1)

The `optimize-pdf` script relies on Ghostscript >= 9.10. Otherwise, it may actually make the PDF larger. Also, you

should only consider using it if the file size of the original PDF is > 5MB.

If a file is found with the extension `.pdfmark` and the same rootname as the input file, it is used to add metadata to the generated PDF document. This file is necessary to preserve the document metadata since Ghostscript will otherwise drop it. That's why Asciidoctor PDF always creates this file in addition to the PDF.

Test a Pull Request

To test a pull request (PR), you first need to fetch the branch that contains the change and switch to it. The steps below are covered in detail in the GitHub help.

Let's assume you want to test PR 955. Here's how you fetch and switch to it:

```
$ git fetch origin pull/955/head:pr-955-review
  git checkout pr-955-review
```

Make sure you replace the number with the number of the PR you want to test.

In case any dependencies have changed, you should run the `bundle` command again:

Now you can run the application as modified by the PR:

```
$ bundle exec asciidoctor-pdf /path/to/sample.adoc
```

To switch back to master, just type:

Asciidoctor PDF was written by Dan Allen and Sarah White of OpenDevise Inc. on behalf of the Asciidoctor Project.

Copyright