a Ruby tool to deface PDF documents — Edit

146 commits 1 branch 4 releases 1 contributor

Branch: master podoff / +

jmettraux add file -i/-I output before qpdf --check output Latest commit 0bc0ba9 an hour ago

bin	fill bin/podoff readme section	12 days ago
lib	1.2.0 pre	2 hours ago
pdfs	ensure the spec start pdfs are valid	11 hours ago
scratch	toy with Zlib::Deflate	17 days ago
spec	add file -i/-I output before qpdf --check output	an hour ago
tmp	adapt t.rb to new api	19 days ago
.gitignore	implement Obj#prepend_text	21 days ago
.travis.yml	try the "apt" Travis addon	13 hours ago
CHANGELOG.txt	1.2.0 pre	2 hours ago
Gemfile	initial commit	22 days ago
Gemfile.lock	force podoff 1.2.0 in Gemfile.lock	2 hours ago
LICENSE.txt	initial commit	22 days ago
README.md	drop Podoff::Obj#page_number	7 days ago
Rakefile	initial commit	22 days ago
podoff.gemspec	initial commit	22 days ago

README.md

podoff

A Ruby tool to deface PDF documents.

Uses "incremental updates" to do so.

Podoff is used to write over PDF documents. Those documents should first be uncompressed (and recompressed) (how? see below)

```
require 'podoff'

d = Podoff.load('d2.pdf')
  # load my d2.pdf

fo = d.add_base_font('Helvetica')
  # make sure the document knows about "Helvetica"
  # (one of the base 13 or 14 fonts PDF readers know about)


pa = d.page(1)
  # grab first page of the document

pa.insert_font('/MyHelvetica', fo)
```

```
    # link "MyHelvetica" to the base font above for this page

  st =
    d.add_stream {
      tf '/MyHelvetica', 12 # Helvetica size 12
      bt 100, 100, "#{Time.now} stamped via podoff" # text at bottom left
    }

  pa.insert_content(st)
    # add content to page

  d.write('d3.pdf')
    # write stamped document to d3.pdf
```

For more about the podoff "api", read "how I use podoff".

If you're looking for serious libraries, look at

- https://github.com/payrollhero/pdf_tempura
- https://github.com/prawnpdf/prawn-templates

preparing documents for use with podoff

Podoff is naive and can't read xref tables in object streams. You have to work against PDF documents that have vanilla xref tables. Qpdf to the rescue.

Given a doc0.pdf you can produce such a document by doing:

```
qpdf --object-streams=disable doc0.pdf doc1.pdf
```

doc1.pdf is now ready for overwriting with podoff.

qpdf has rewritten the PDF, extracting the xref table but keeping the streams compressed.

bin/podoff

`bin/podoff` is a command-line tool for to preparing/check PDFs before use.

```
$ ./bin/podoff -h

Usage: ./bin/podoff [option] {fname}

    -o, --objs                       List objs
    -w, --rewrite                    Rewrite
    -s, --stamp                      Apply time stamp at bottom of each page
    -r, --recompress                 Recompress
    --version                        Show version
    -h, --help                       Show this message
```

`--recompress` is mostly an alias for `qpdf --object-streams=disable in.pdf out.pf`

`--stamp` is used to check whether podoff can add a time stamp on each page of an input PDF.

how I use podoff

In the application which necessitated the creation of podoff, there are two PDF to generate from time to time.

I keep those two PDFs in memory.

```
# lib/myapp/pdf.rb
```

```
require 'podoff'

module MyApp::Pdf

  DOC0 = Podoff.load('pdf_templates/d0.pdf')
  DOC1 = Podoff.load('pdf_templates/d1.pdf')

  def generate_doc0(data, path)

    d = DOC0.dup # shallow copy of the document
    d.add_fonts

    pa2 = d.page(2)
    st = d.add_stream # open stream...

    st.font 'MyHelv', 12 # font is an alias to tf
    st.text 100, 100, data['customer_name']
    st.text 100, 80, data['customer_phone']
    st.text 100, 60, data['date'] if data['date']
      # fill in customer info on page 2

    pa2.insert_content(st) ... close stream (yes, you can use a block too)

    pa3 = d.page(3)
    pa3.insert_content(d.add_stream { check 52, 100 }) if data['discount']
      # a single check on page 3 if the customer gets a discount

    d.write(path)
  end

  # ...
end

module Podoff # adding a few helper methods to the podoff classes

  class Document

    # Makes sure Helvetica and ZapfDingbats are available
    # on each page of the document
    #
    def add_fonts

      fo0 = add_base_font('/Helvetica')
      fo1 = add_base_font('/ZapfDingbats')

      pages.each { |pa|
        pa = re_add(pa)
        pa.insert_font('/MyHelv', fo0)
        pa.insert_font('/MyZapf', fo1)
      }
    end
  end

  class Stream

    # Places a check mark ✓ at x, y
    #
    def check(x, y)

      font = @font            # save current font
      self.tf '/MyZapf', 12   # switch to ZapfDingbats size 12
      self.bt x, y, '3'       # check mark
      @font = font            # get back to saved font
    end
  end
end
```

The documents are kept in memory, as generation request comes, the get duplicated, incrementally updated and the filled documents are written to disk. The duplication doesn't copy the whole document file, only the references to the "obj" in the document get copied.

Podoff::Document

```
class Podoff::Document

  def self.load(path, encoding='iso-8859-1')
    # Podoff.load(path, encoding) is a shortcut to this method

  def dup
    # Makes a shallow copy of the document

  def add_base_font(name)
    # Given a name in the base 13/14 fonts readers are supposed to know,
    # ensures the document has access to the font.
    # Usually "Helvetica" or "ZapfDingbats".

  def pages
    # Returns an array of all the objs that are pages

  def page(index)
    # Starts at 1, returns a page obj. Understands negative indexes, like
    # -1 for the last page.

  def add_stream(src=nil, &block)
    # Prepares a new obj with a stream
    # If src is given places the src string in the stream.
    # If a block is given executes the block in the context of the
    # Podoff::Stream instance.
    # If no src and no block, simply returns the Podoff::Stream wrapped inside
    # of the new obj (see example code above)

  def re_add(obj_or_ref)
    # Given an obj or a ref (like "1234 0") to an obj, copies that obj
    # and re-adds it to the document.
    # This is necessary for the incremental updates podoff uses, if you add
    # an obj to the Contents list of a page, you have to add it to the
    # re-added page, not directly to the original page.

  def write(path=:string)
    # Writes the document, with incremental updates to a file given by its path.
    # If the path is :string, will simply return the string containing the
    # whole document

  def rewrite(path=:string)
    # Like #write, but squashes the incremental updates in the document.
    # Takes more time and memory and might fail (remember, podoff is very
    # naive (as his author is)). Test with care...

  #
  # a bit lower-level...

  def objs
    # returns the hash { String/obj_ref => Podoff::Obj/obj_instance }
```

Podoff::Obj

A PDF document is mostly a hierarchy of `obj` elements. `Podoff::Obj` points to such elements (see `Podoff::Document#objs`).

```
class Podoff::Obj

  def insert_font(font_nick, font_obj_or_ref)
  def insert_contents(obj_or_ref)
```

Podoff::Stream

TODO

```
class Podoff::Stream

  def tf(font_name, font_size)
  alias :font :tf

  def bt(x, y, text)
  alias :text :bt
```

disclaimer

The author of this tool/library have no link whatsoever with the authors of the sample PDF documents found under `pdfs/`. Those documents have been selected because they are representative of the PDF forms podoff is meant to ~~deface~~fill.

known bugs

- podoff parsing is naive, documents that contain uncompressed streams with "endobj", "startxref", "/Root" will disorient podoff
- completely candid about encoding (only used it for British English documents so far)

links

- http://qpdf.sourceforge.net/ source: https://github.com/qpdf/qpdf
- http://www.slideshare.net/ange4771/advanced-pdf-tricks

LICENSE

MIT, see LICENSE.txt