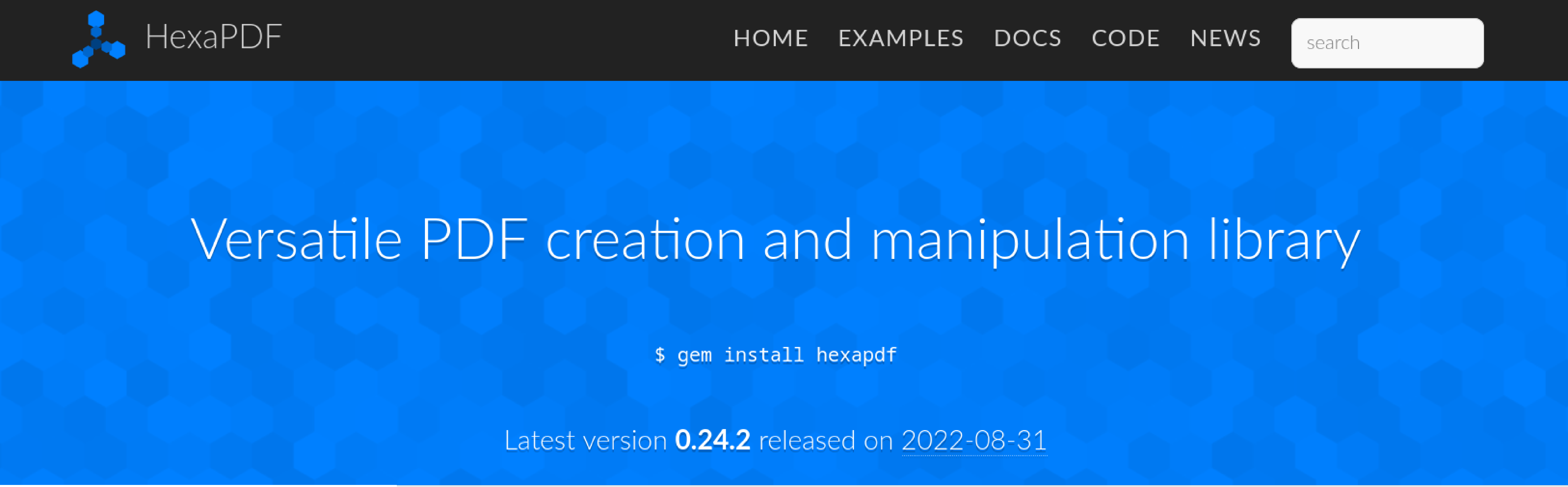


Versatile PDF creation and manipulation library

```
$ gem install hexapdf
```

Latest version **0.24.2** released on 2022-08-31

working together!

PDF Days Europe 2022 | Berlin

Implementing a PDF Library in Ruby

Design and implementation decisions to create a fully-featured, fast and memory-efficient PDF library

Table of Contents

- Why Ruby?
- Why HexaPDF?
- Parsing
- Object Model
- Serializing
- Encryption
- Document Layout Engine
- Optimizations
- General Design

working together!

PDF Days Europe 2022 | Berlin

Why Ruby?

- Language of choice for side projects
- Fast development/feedback loop
- **No existing, feature-complete, pure Ruby PDF library**

Why HexaPDF?

- Scratching an itch (PDF generation library Prawn not good enough)
- Desire to implement a complex, publicly available specification
- Provide high-quality library and **dual-license AGPL+commercial to sustain development**

HexaPDF Code Example

```ruby
require 'hexapdf'

doc = HexaPDF::Document.new
doc.trailer.info[:CreationDate] = Time.now
doc.trailer.info[:Title] = "My Hello World"
canvas = doc.pages.add.canvas
canvas.font('Helvetica', size: 100)
canvas.text("Hello World!", at: [20, 400])
doc.write("hello-world.pdf", optimize: true)
```

Parsing

- Usual separation between tokenizer and parser
- Tokenizer also responsible for parsing whole PDF objects
- **StringScanner and regular expressions used for tokenization**, not byte-by-byte

Parsing - StringScanner Usage

```ruby
# Parses the hex string at the current position.
#
# See: PDF1.7 s7.3.4.3
def parse_hex_string
  @ss.pos += 1                    # @ss is the StringScanner object
  data = scan_until(/(?=>)/)
  unless data
    raise HexaPDF::MalformedPDFError.new("Unclosed hex string found", pos: pos)
  end
  @ss.pos += 1
  data.tr!(WHITESPACE, "")
  [data].pack('H*')
end
```

Parsing - Continued

- First iteration done like described in the specification, then added work-arounds for invalid PDFs
- Now capable of parsing a wide array of invalid PDFs

Object Model

- Many implementations create custom classes for PDF object types
- **HexaPDF uses built-in Ruby classes** where possible for performance, lower memory usage and ease of use
- **Nearly perfect mapping** between PDF object types and Ruby's built-in classes

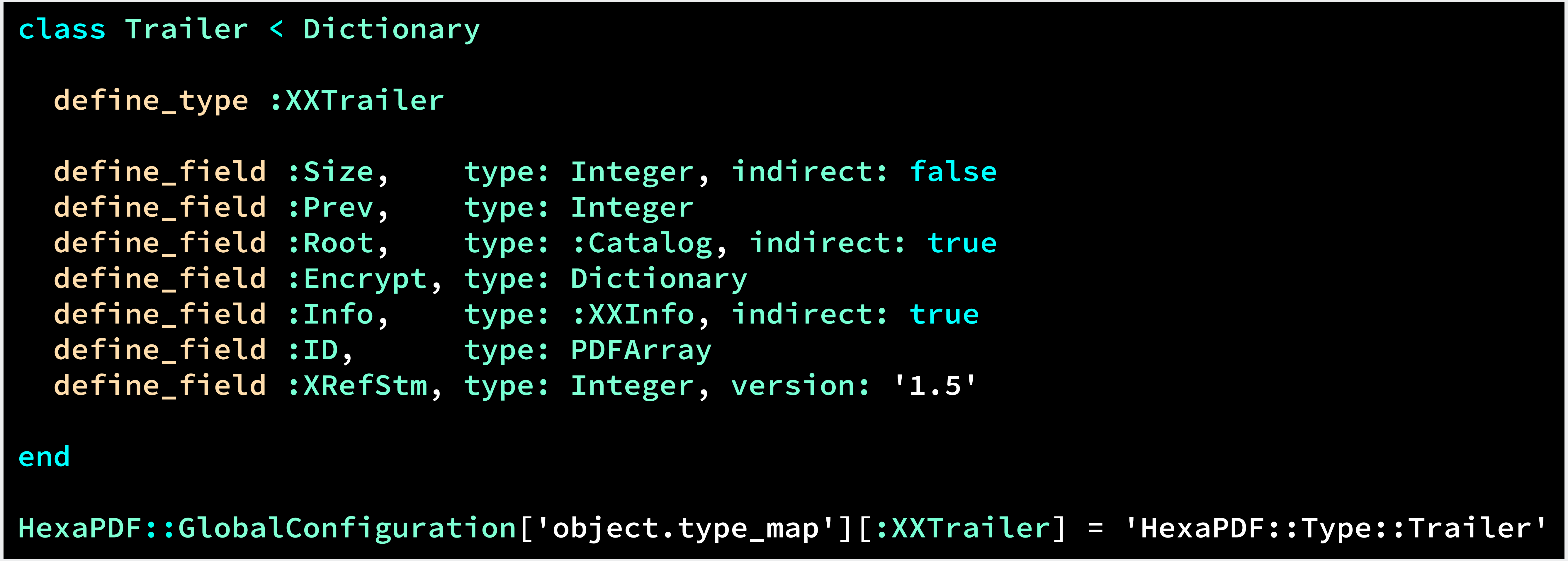
Object Model - Type Mapping

Boolean	true/false
Numeric	Integer, Float
String	String
Name	Symbol
Null	nil
Array	Array, HexaPDF::PDFArray
Dictionary	Hash, HexaPDF::Dictionary
Indirect Objects	HexaPDF::Object
Stream	HexaPDF::Stream

Evolution of Object Model

- Use custom `HexaPDF::Dictionary` class in addition to the built-in Hash class for dictionaries
- Allows **automatic reference resolving** on access
- PDF types like the document catalog as subclasses with **field definitions**

PDF Type Implementation



```
class Trailer < Dictionary

  define_type :XXTrailer

  define_field :Size,    type: Integer, indirect: false
  define_field :Prev,    type: Integer
  define_field :Root,    type: :Catalog, indirect: true
  define_field :Encrypt, type: Dictionary
  define_field :Info,    type: :XXInfo, indirect: true
  define_field :ID,      type: PDFArray
  define_field :XRefStm, type: Integer, version: '1.5'

end

HexaPDF::GlobalConfiguration['object.type_map'][:XXTrailer] = 'HexaPDF::Type::Trailer'
```

Dictionary Fields Usages

- Return **default values** for unset fields on access
- Field information is used for basic **object validation**
- **Automatic conversion** of values based on the field type (e.g. PDF dates but also specific type classes)

Memory Usage vs Ease-of-use

- Auto-converting dictionaries is a trade-off between lower memory usage and ease-of-use
- Core data (object number, generation number, value, stream) is put in `HexaPDF::PDFData` object
- Data object is wrapped by one or more `HexaPDF::Object` instances

Auto-converting Dictionaries

- Automatically convert loaded or added hashes/dictionaries to specific type classes
- Conversion is based on the `/Type` and `/Subtype` entries as well as the required fields
- Problem: False positives, e.g. artifact property list containing `<</Type /Page>>`

Serializing

- Any Ruby object can be serialized, serializer is extendable
- Compact serialization representation by default

Serializing Strings

```
STRING_ESCAPE_MAP = {"(" => "\\(", ")" => "\\)", "\\" => "\\\\", "\r" => "\\r"}.freeze

def serialize_string(obj)
  obj = if @encrypter && @object.kind_of?(HexaPDF::Object) && @object.indirect?
          encrypter.encrypt_string(obj, @object)
        elsif obj.encoding != Encoding::BINARY
          if obj.match?(/[^ -~\t\r\n]/)
            "\xFE\xFF".b << obj.encode(Encoding::UTF_16BE).force_encoding(Encoding::BINARY)
          else
            obj.b
          end
        else
          obj.dup
        end
  obj.gsub!(/[()\\\r]/n, STRING_ESCAPE_MAP)
  "(#{obj})"
end
```

Encryption

- Done at a later stage, kind of "tucked" on but in a good way
- Loading encrypted objects is nearly orthogonal to parsing, only the object loader is amended
- Serialization class adapted to provide encryption

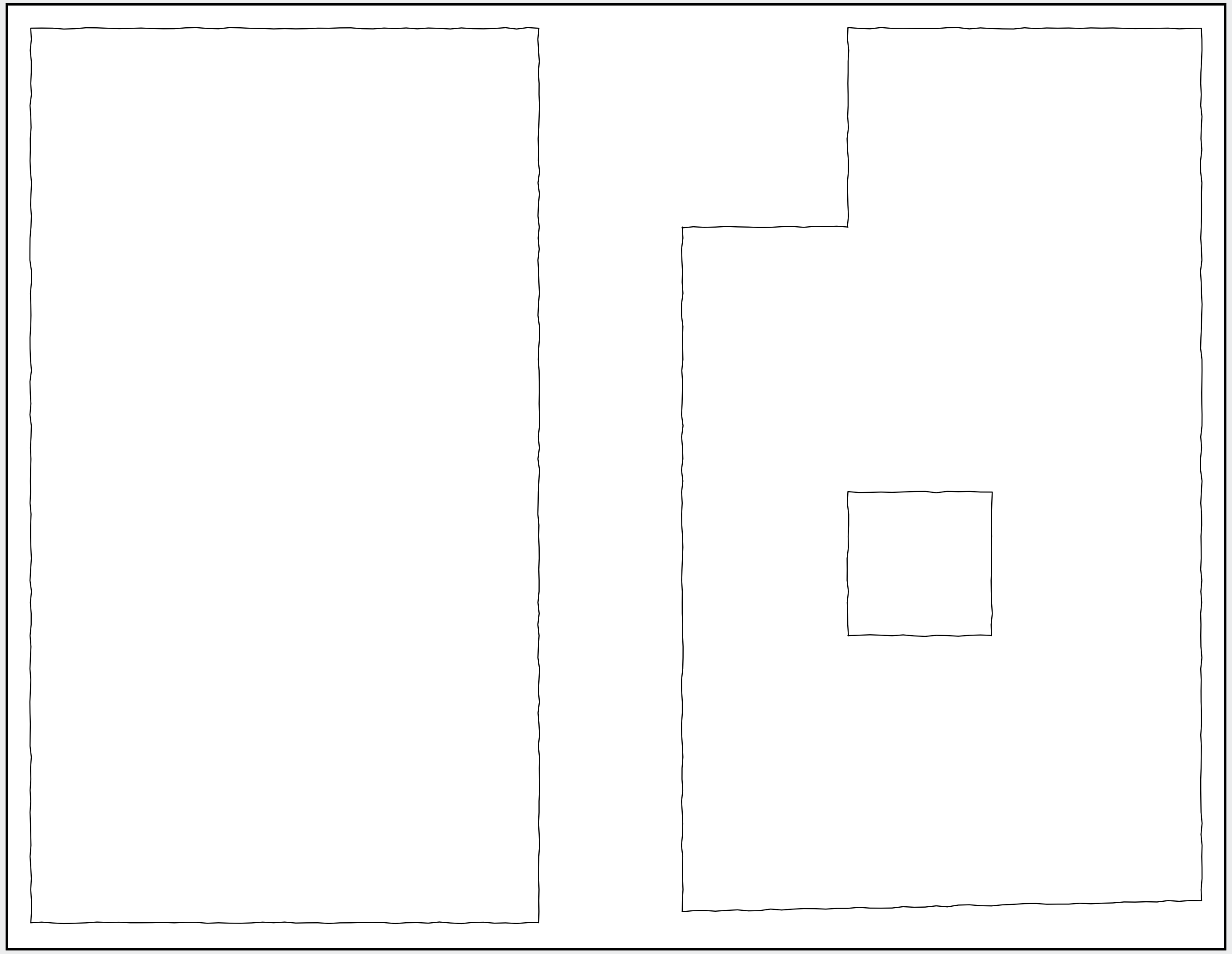
Decryption of Loaded Object

```
document.revisions.each do |r|
  loader = r.loader
  r.loader = lambda do |xref_entry|
    obj = loader.call(xref_entry)
    xref_entry.compressed? ? obj : handler.decrypt(obj)
  end
end
```

Document Layout Engine

- PDF needs explicit glyph positions, conforming **writer does all the layout**
- HexaPDF's layout engine is written from scratch, based on the concept of **frames** and **boxes**
- Frames provide an area to place boxes and know where and how the next box can be placed
- The frame area, its shape, is usually rectangular but can generally be a set of rectilinear polygons

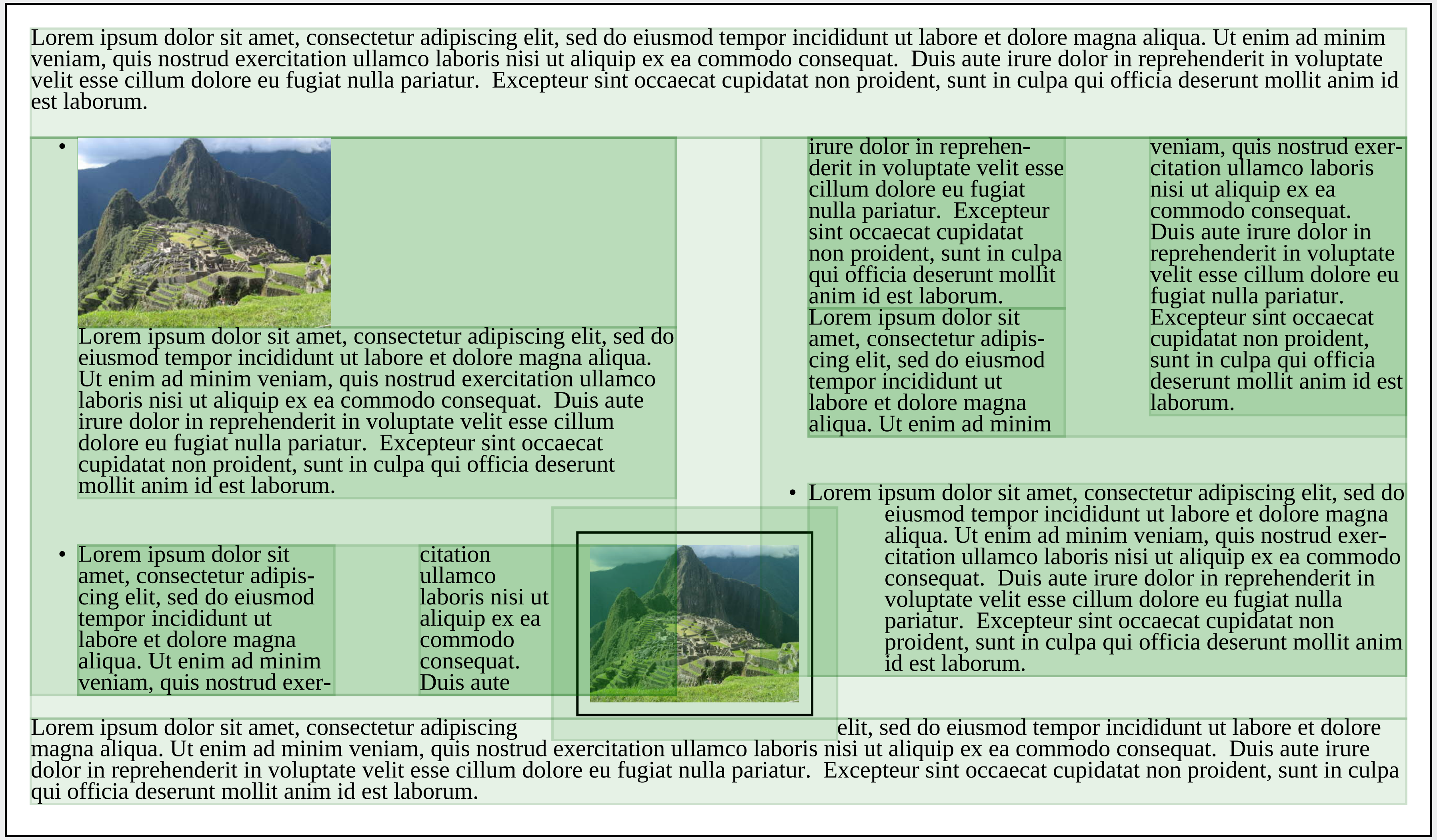
Document Layout - Frame



Document Layout Cont.

- Boxes are responsible for three things: **fitting** their contents, **splitting** the box if necessary and **drawing** their contents
- A box may be fitted into a rectangular area or use the frame's shape to flow its contents
- The text layout engine can **flow text inside a set or arbitrary polygons**

Document Layout - Example



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

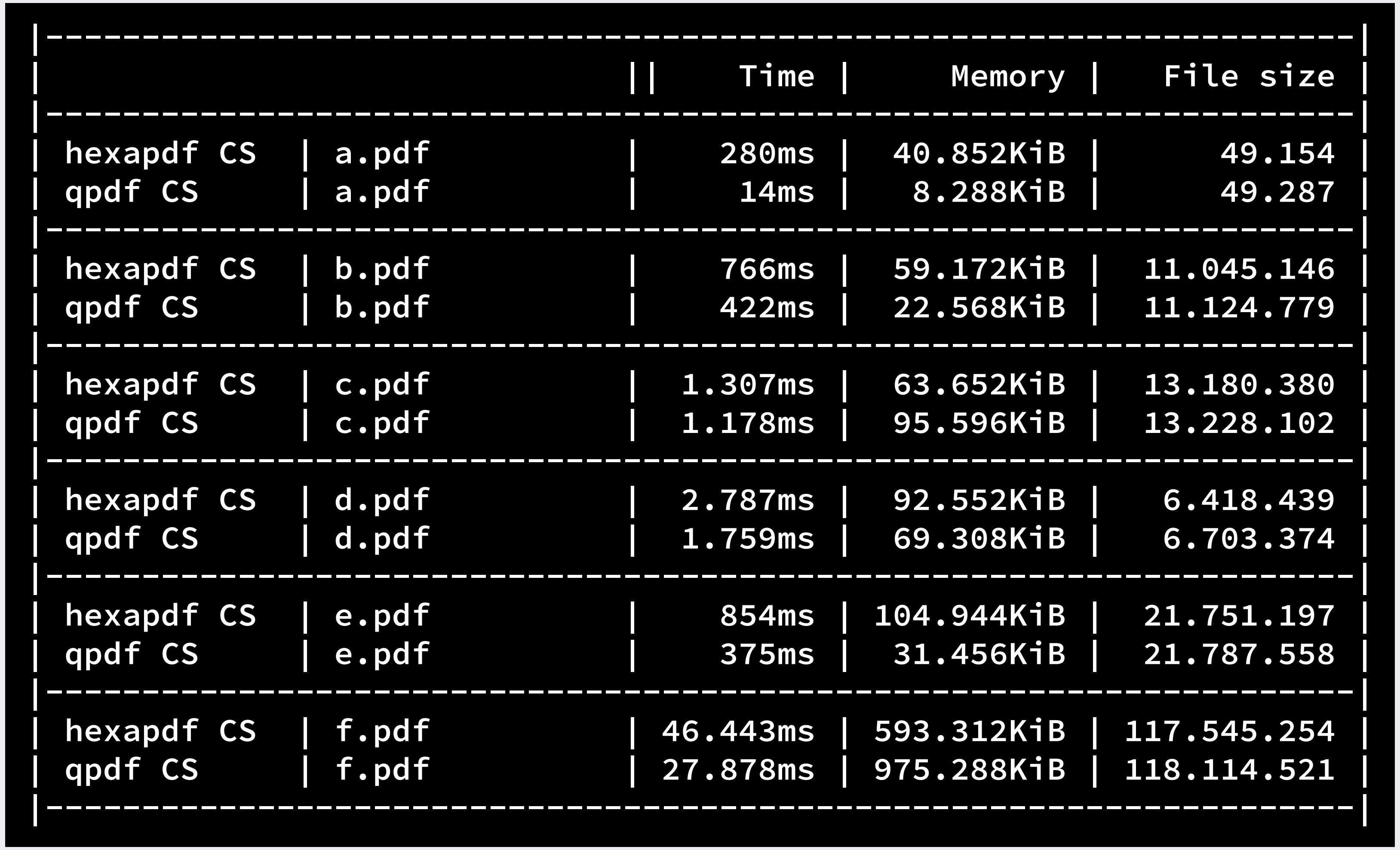
- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
- Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Optimizations

- Stream filters implemented via **fiber pipeline** (allow processing streams in chunks, only apply necessary filters)
- **Optimize size of serialized output** (smaller files and faster operation due to less serialization)
- **Optimize text output** (use most compact representation possible)

Size Optimization Benchmark



```
|--------------------------------------------------------------------|
|                              ||    Time |     Memory |   File size |
|--------------------------------------------------------------------|
| hexapdf CS  | a.pdf          |    280ms |  40.852KiB |      49.154 |
| qpdf CS     | a.pdf          |     14ms |   8.288KiB |      49.287 |
|--------------------------------------------------------------------|
| hexapdf CS  | b.pdf          |    766ms |  59.172KiB |  11.045.146 |
| qpdf CS     | b.pdf          |    422ms |  22.568KiB |  11.124.779 |
|--------------------------------------------------------------------|
| hexapdf CS  | c.pdf          |  1.307ms |  63.652KiB |  13.180.380 |
| qpdf CS     | c.pdf          |  1.178ms |  95.596KiB |  13.228.102 |
|--------------------------------------------------------------------|
| hexapdf CS  | d.pdf          |  2.787ms |  92.552KiB |   6.418.439 |
| qpdf CS     | d.pdf          |  1.759ms |  69.308KiB |   6.703.374 |
|--------------------------------------------------------------------|
| hexapdf CS  | e.pdf          |    854ms | 104.944KiB |  21.751.197 |
| qpdf CS     | e.pdf          |    375ms |  31.456KiB |  21.787.558 |
|--------------------------------------------------------------------|
| hexapdf CS  | f.pdf          | 46.443ms | 593.312KiB | 117.545.254 |
| qpdf CS     | f.pdf          | 27.878ms | 975.288KiB | 118.114.521 |
|--------------------------------------------------------------------|
```

General Design

- Make everything possible using a low-level interface
- Iteratively add classes for the PDF types (for automatic validation and value conversion)
- Add convenience methods for easier usage (e.g. manipulating the page tree)

General Design Cont.

- Provide convenience methods on the class where they are most useful
- If no class fits, create a new one and allow easy access via the main document object
- Make everything extendable and exchangable where possible and useful

working together!

PDF Days Europe 2022 | Berlin

Thank you!