Pdfimpose Documentation

Release 2.5.0

Louis Paternault

Mar 05, 2024

CONTENTS

1 Printing 3

2 Contents 5

3 Examples 17

4 See also 19

5 Download and install 21

6 Indices and tables 23

Python Module Index 25

Index 27

pdfimpose is a library and a command line program to impose a PDF document. According to Wikipedia, "imposition consists in the arrangement of the printed product's pages on the printer's sheet, in order to obtain faster printing, simplify binding and reduce paper waste".

Warning:

- I am not a printing expert (I am not even sure I deserve to be called a printing hobbyist).
- English is not my first language.
- The few things I inaccurately know about printing, I know them in my first language.

Those are three reasons why this documentation might be sometimes unclear. If you have time to spare, I would really appreciate some proofreading.

CHAPTER

ONE

PRINTING

- When `pdfimpose` has to guess the size of the output paper, it uses the A4 format. This is an (arbitrary) implementation detail, and might change in future releases.
- When printing an imposed PDF, it shall be printed two-sided, the binding edge on the left or right.

CHAPTER

TWO

CONTENTS

2.1 Command line

This module includes a command line client: *pdfimpose*, which can be used to impose a PDF document, using one of the *imposition schemas*.

Contents

- *Schemas*
- *Configuration file*

2.1.1 Schemas

You can impose file using any schemas with the following command line:

```
pdfimpose SCHEMA foo.pdf
```

For instance, to impress your A5 document using *saddle stitch* (like in magazines), use:

```
pdfimpose saddle foo.pdf
```

Each schema have different options. Use `pdfimpose SCHEMA --help` for more information.

2.1.2 Configuration file

Subcommand `apply` can be used to store options in a configuration file:

```
pdfimpose apply [-h] [--schema SCHEMA] [CONF] [PDF ...]
```

- `CONF` is a configuration file, in *yaml* format (see below);
- `PDF` is the file(s) to process;
- `SCHEMA` is the imposition schema to use.

Those three arguments are optional: `pdfimpose apply` is a valid command line:

- If `CONF` is missing, a configuration file is searched:
  - `pdfimpose.cfg` or `.pdfimpose.cfg`, in the current working directory;

  - the same files, in the parent directory, or grand-parent directory, or…;
  - the same files, in `~/.config`;
  - the same files, in the home directory;
  - `/etc/pdfimpose.cfg` (depending on the operating system).
- If `PDF` is missing, it is read from the configuration file (section `general`, option `files`).
- If `SCHEMA` is missing, it is read from the configuration file (section `general`, option `schema`).

For instance, calling `pdfimpose apply foo.cfg`, where `foo.cfg` contains:

```
[general]
schema = hardcover
files = foo.pdf bar.pdf

[hardcover]
imargin = 1cm
omargin = .5cm
```

is equivalent to the following command line:

```
pdfimpose hardcover --imargin 1cm --omargin .5cm foo.pdf bar.pdf
```

2.2 Library

Most of the classes and function of `pdfimpose` are not *private* (their name do not start with and underscore), but the only ones being maintained are the following ones. Use the others at your own risk!

Note: The description of each imposition schemas would be a lot more understandable with a nice drawing or video. I cannot do any of those. If you can, and you have some time to spare, it would be very welcome!

2.2.1 `pdfimpose.schema`

Common classes and function to different imposition schemas.

Each submodule provides:

- a class `SCHEMAImpositor`, where:
  - its `SCHEMAImpositor.__init__()` method takes the schema arguments (which are more or less the same ones as the corresponding command line subcommand);
  - its `SCHEMAImpositor.impose()` method take the source and destination file names as arguments, and performs the imposition;
- a function `impose()`, which is barely more than a wrapper to the aforementionned class.

Margins

class `pdfimpose.schema.`**`Margins`**(*left: float = 0*, *right: float = None*, *top: float = None*, *bottom: float = None*)

Left, right, top, bottom margins.

- If the constructor has only one argument, all four margins are equal to this value.
- If the constructor has no argument, all four margins are 0.

Page

class `pdfimpose.schema.`**`Page`**(*number: int*, *rotate: int = 0*, *left: int = 0*, *right: int = 0*, *top: int = 0*, *bottom: int = 0*)

A virtual page: a page number, a rotation, and margins.

Matrix

class `pdfimpose.schema.`**`Matrix`**(*pages: list[list[~pdfimpose.schema.Page]] = <factory>*, *rotate: dataclasses.InitVar[int] = 0*)

Imposition matrix.

This matrix does not contain actual pages, but an array of which source page numbers should go where on one output page.

AbstractImpositor

class `pdfimpose.schema.`**`AbstractImpositor`**(*last: int = 0*, *omargin: ~pdfimpose.schema.Margins | str | ~numbers.Real | ~decimal.Decimal = <factory>*, *mark: list[str] = <factory>*)

Perform imposition of source files onto output file.

This is an abstract method, with common methods, to be inherited by imposition schemas.

2.2.2 `pdfimpose.schema.cards`

Cut as flash cards (question on front, answer on back).

This schema can be used when you want to print flash cards:

- your source PDF is a list of (let's say) A6 pages: Question 1, Answer 1, Question 2, Answer 2, Question 3, Answer 3… Note that this behavior can be changed with option –back.
- you want to print those questions and answer on an A4 sheet of paper, and cut it to get your flash cards (questions on front, answers on back).

Example:
`source`, `destination`.

`CardsImpositor`

class `pdfimpose.schema.cards.`**`CardsImpositor`**(*last: int = 0, omargin: ~pdfimpose.schema.Margins | str | ~numbers.Real | ~decimal.Decimal = <factory>, mark: list[str] = <factory>, imargin: float = 0, signature: tuple[int] = (0, 0), back: str = ''*)

Perform imposition of source files, with the 'card' schema.

`impose()`

`pdfimpose.schema.cards.`**`impose`**(*files, output, *, imargin=0, omargin=0, mark=None, signature=None, size=None, back=''*)

Perform imposition of source files into an output file, to be cut as flash cards.

Parameters

- **files** (*list[str]*) – List of source files (as strings or `io.BytesIO` streams). If empty, reads from standard input.
- **output** (*str*) – List of output file.
- **omargin** (*float*) – Output margin, in pt. Can also be a *pdfimpose.schema.Margins* object.
- **imargin** (*float*) – Input margin, in pt.
- **mark** (*list[str]*) – List of marks to add. Only crop marks are supported (`mark=['crop']`); everything else is silently ignored.
- **signature** (*tuple[int]*) – Layout of source pages on output pages. For instance, `(2, 3)` means that each output page will contain 2 columns and 3 rows of source pages. Incompatible with option *size*.
- **size** (*str | tuple[float]*) – Size of the output page. Signature is computed to fit the page. This option is incompatible with *signature*.
- **back** (*Optional[str]*) – Back sides of cards. See –back help for more information.

2.2.3 `pdfimpose.schema.copycutfold`

Print pages, to be cut and folded, and eventually bound, to produce multiple books.

You want to print and bind several copies of a tiny A7 book. Those books are made with A6 sheets (when you open the book, you get two A7 pages side-by-side, which is A6). Since you can fit four A6 pages on an A4 page, this means that you can print four books at once.

To use this schema (without option –group):

- print your imposed file, two-sided;
- cut the stack of paper, to get several stacks (four in the example above);
- fold (once) and bind each stack of paper you got, separately;
- voilà! You now have several copies of your book.

With option –group=3 (for instance), repeat the step above for every group of three sheets. You get several signatures, that you have to bind together to get a proper book.

Example:
`source`, `destination`.

`CopyCutFoldImpositor`

class pdfimpose.schema.copycutfold.**CopyCutFoldImpositor**(*last: int = 0, omargin: ~pdfimpose.schema.Margins | str | ~numbers.Real | ~decimal.Decimal = <factory>, mark: list[str] = <factory>, bind: str = 'left', creep: ~typing.Callable[[int], float] = <function nocreep>, imargin: str | ~numbers.Real | ~decimal.Decimal = 0, signature: tuple[int] = (0, 0), group: int = 0*)

Perform imposition of source files, with the 'copycutfold' schema.

`impose()`

pdfimpose.schema.copycutfold.**impose**(*files*, *output*, ***, *imargin=0*, *omargin=0*, *last=0*, *mark=None*, *signature=None*, *size=None*, *bind='left'*, *creep=<function nocreep>*, *group=0*)

Perform imposition of source files into an output file, using the copy-cut-fold schema.

Parameters

- **files** (*list[str]*) – List of source files (as strings or `io.BytesIO` streams). If empty, reads from standard input.
- **output** (*str*) – List of output file.
- **omargin** (*float | numbers.Real | decimal.Decimal | Margins*) – Output margin. It can be: a `numbers.Real` or `decimal.Decimal`` (unit is pt), a `Margins` object, a `str`, to be parsed by `papersize.parse_length()`.
- **imargin** (*float | numbers.Real | decimal.Decimal*) – Input margin. Same types and meaning as *omargin* (excepted that `Margins` objects is not accepted).
- **last** (*int*) – Number of last pages (of the source files) to keep at the end of the output document. If blank pages were to be added to the source files, they would be added before those last pages.
- **mark** (*list[str]*) – List of marks to add. Only crop marks are supported (*mark=['crop']*); everything else is silently ignored.
- **signature** (*tuple[int]*) – Layout of source pages on output pages. For instance `(2, 3)` means: the printed sheets are to be cut in a matrix of 2 horizontal sheets per 3 vertical sheets. This option is incompatible with *size*.
- **size** (*str | tuple[float]*) – Size of the output page. Signature is computed to fit the page. This option is incompatible with *signature*.
- **bind** (*str*) – Binding edge. Can be one of *left*, *right*, *top*, *bottom*.
- **creep** (*function*) – Function that takes the number of sheets in argument, and return the space to be left between two adjacent pages.
- **group** (*int*) – Group sheets before cutting them. See help of command line –group option for more information.

2.2.4 `pdfimpose.schema.cutstackfold`

Print pages, to be cut, stacked, folded, and eventually bound.

Example: You want to print and bind one single tiny A7 book. This book is made with A6 sheets (when you open the book, you get two A7 pages side-by-side, which is A6). Since you can fit four A6 pages on an A4 page, this means that you can print four A6 sheets on one A4 sheet.

To use this schema (without using option –group):

- print your imposed file, two-sided;
- cut the stack of paper, to get several stacks (four in the example above);
- stack the several stacks you got on top of each other (take care to keep the pages in the right order);
- fold and bind the stack of paper you got;
- voilà! You now have a shiny, tiny book.

With option –group=3 (for instance), repeat the step above for every group of three sheets. You get several signatures, that you have to bind together to get a proper book.

Example:
`source`, `destination`.

`CutStackFoldImpositor`

class `pdfimpose.schema.cutstackfold.`**`CutStackFoldImpositor`**(*last: int = 0, omargin: ~pdfimpose.schema.Margins | str | ~numbers.Real | ~decimal.Decimal = <factory>, mark: list[str] = <factory>, bind: str = 'left', creep: ~typing.Callable[[int], float] = <function nocreep>, imargin: str | ~numbers.Real | ~decimal.Decimal = 0, signature: tuple[int] = (0, 0), group: int = 0*)

Perform imposition of source files, with the 'cutstackfold' schema.

`impose()`

`pdfimpose.schema.cutstackfold.`**`impose`**(*files*, *output*, ***, *imargin=0*, *omargin=0*, *last=0*, *mark=None*, *signature=None*, *size=None*, *bind='left'*, *creep=<function nocreep>*, *group=0*)

Perform imposition of source files into an output file, using the cut-stack-bind schema.

Parameters

- **`files`** (*`list[str]`*) – List of source files (as strings or `io.BytesIO` streams). If empty, reads from standard input.
- **`output`** (*`str`*) – List of output file.
- **`omargin`** (*`float`*) – Output margin, in pt. Can also be a `Margins` object.
- **`imargin`** (*`float`*) – Input margin, in pt.

- **last** (*int*) – Number of last pages (of the source files) to keep at the end of the output document. If blank pages were to be added to the source files, they would be added before those last pages.
- **mark** (*list[str]*) – List of marks to add. Only crop marks are supported (*mark=['crop']*); everything else is silently ignored.
- **signature** (*tuple[int]*) – Layout of source pages on output pages. For instance `(2, 3)` means: the printed sheets are to be cut in a matrix of 2 horizontal sheets per 3 vertical sheets. This option is incompatible with *size*.
- **size** (*str | tuple[float]*) – Size of the output page. Signature is computed to fit the page. This option is incompatible with *signature*.
- **bind** (*str*) – Binding edge. Can be one of *left*, *right*, *top*, *bottom*.
- **creep** (*function*) – Function that takes the number of sheets in argument, and return the space to be left between two adjacent pages (that is, twice the distance to the spine).
- **group** (*int*) – Group sheets before cutting them. See help of command line –group option for more information.

2.2.5 `pdfimpose.schema.onepagezine`

A one-page fanzine, with a poster on the back.

On this schema, you get an 8 pages book which, once unfolded, gives a poster on the back (see some photos).

This command only perform imposition of the front of your fanzine. It is your job to print the poster on the back.

Example:
`source`, `destination`.

`OnePageZineImpositor`

class `pdfimpose.schema.onepagezine.`**`OnePageZineImpositor`**(*last: int = 0*, *omargin: ~pdfimpose.schema.Margins | str | ~numbers.Real | ~decimal.Decimal = <factory>*, *mark: list[str] = <factory>*, *bind: str = 'left'*)

Perform imposition of source files, with the 'one-page-zine' schema.

See *http://experimentwithnature.com/03-found/experiment-with-paper-how-to-make-a-one-page-zine/*.

`impose()`

`pdfimpose.schema.onepagezine.`**`impose`**(*files*, *output*, ***, *omargin=0*, *last=0*, *mark=None*, *bind='left'*)

Perform imposition of source files into an output file, to be printed as a "one page zine".

Parameters

- **files** (*list[str]*) – List of source files. If empty, reads from standard input.
- **output** (*str*) – List of output file.
- **omargin** (*number.Real | Margins | decimal.Decimal | str*) – Output margin, as: a number `number.Real` or `decimal.Decimal`, a `Margins` object, or a `str`, that is to be parsed by `papersize.parse_length()`.

- **last** (*int*) – Number of last pages (of the source files) to keep at the end of the output document. If blank pages were to be added to the source files, they would be added before those last pages.
- **mark** (*list[str]*) – List of marks to add. Only crop marks are supported (*mark=['crop']*); everything else is silently ignored.
- **bind** (*str*) – Binding edge. Can be one of *left*, *right*, *top*, *bottom*.

See *http://experimentwithnature.com/03-found/experiment-with-paper-how-to-make-a-one-page-zine/*.

2.2.6 pdfimpose.schema.hardcover

Hardcover binding (done in books, like dictionaries)

Hardcover binding is the schema used to print *real* books (novels, dictionnaries, etc.): several destination pages are printed on a single, big, sheet of paper, which is folded, and cut. Those booklets are stacked onto each other, and bound together, to make your book.

To use this schema (without option –group, or with –group=1):

- print your imposed PDF file, two-sided;
- separately fold each sheet of paper;
- stack them;
- bind them.

With option –group=3 (for instance), repeat the step above for every group of three sheets. You get several signatures, that you have to bind together to get a proper book.

Example:
`source`, `destination`.

HardcoverImpositor

class pdfimpose.schema.hardcover.**HardcoverImpositor**(*last: int = 0, omargin: ~pdfimpose.schema.Margins | str | ~numbers.Real | ~decimal.Decimal = <factory>, mark: list[str] = <factory>, folds: str = None, imargin: float = 0, bind: str = 'left', group: int = 1*)

Perform imposition of source files, with the 'hardcover' schema.

impose()

pdfimpose.schema.hardcover.**impose**(*files, output, *, folds=None, signature=None, size=None, imargin=0, omargin=0, mark=None, last=0, bind='left', group=1*)

Perform imposition of source files into an output file, to be bound using "hardcover binding".

Parameters

- **files** (*list[str]*) – List of source files (as strings or `io.BytesIO` streams). If empty, reads from standard input.
- **output** (*str*) – List of output file.
- **omargin** (*float*) – Output margin, in pt. Can also be a `Margins` object.

- **imargin** (*float*) – Input margin, in pt.
- **mark** (*list[str]*) – List of marks to add. Only crop marks are supported (*mark=['crop']*); everything else is silently ignored.
- **folds** (*str*) – Sequence of folds, as a string of characters *h* and *v*. Incompatible with *size* and *signature*.
- **size** (*str*) – Size of the destination pages, as a string that is to be parsed by `papersize.parse_papersize()`. This option is incompatible with *signature* and *folds*.
- **signature** (*tuple[int]*) – Layout of source pages on output pages. For instance `(2, 3)` means: the printed sheets are to be cut in a matrix of 2 horizontal sheets per 3 vertical sheets. This option is incompatible with *size* and *folds*.
- **bind** (*str*) – Binding edge. Can be one of *left*, *right*, *top*, *bottom*.
- **last** (*int*) – Number of last pages (of the source files) to keep at the end of the output document. If blank pages were to be added to the source files, they would be added before those last pages.
- **group** (*int*) – Group sheets before folding them. See help of command line –group option for more information.

2.2.7 `pdfimpose.schema.saddle`

Saddle stitch (like in newpapers or magazines)

This schema is used in newspapers or magazines: the sheets are inserted into each other.

To use this schema (with –group=1, or without –group):

- print your imposed PDF file, two-sided;
- **if there is two source pages on each destination page:**
  - fold all your sheets at once;
  - otherwise, separately fold each sheet of paper, and insert them into each other;
- bind.

With option –group=3 (for instance), repeat the step above for every group of three sheets. You get several signatures, that you have to bind together to get a proper book.

Example:
`source`, `destination`.

`SaddleImpositor`

class `pdfimpose.schema.saddle.`**`SaddleImpositor`**(*last: int = 0, omargin: ~pdfimpose.schema.Margins | str | ~numbers.Real | ~decimal.Decimal = <factory>, mark: list[str] = <factory>, folds: str = None, imargin: float = 0, bind: str = 'left', group: int = 1, creep: ~typing.Callable[[int], float] = <function nocreep>*)

Perform imposition of source files, with the 'saddle' schema.

impose()

`pdfimpose.schema.saddle.impose`(*files*, *output*, ***, *folds=None*, *signature=None*, *size=None*, *imargin=0*, *omargin=0*, *mark=None*, *last=0*, *bind='left'*, *creep=<function nocreep>*, *group=1*)

Perform imposition of source files into an output file, to be bound using "saddle stitch".

Parameters

- **files** (*list[str]*) – List of source files (as strings or `io.BytesIO` streams). If empty, reads from standard input.
- **output** (*str*) – List of output file.
- **omargin** (*float*) – Output margin, in pt. Can also be a `Margins` object.
- **imargin** (*float*) – Input margin, in pt.
- **mark** (*list[str]*) – List of marks to add. Only crop marks are supported (*mark=['crop']*); everything else is silently ignored.
- **folds** (*str*) – Sequence of folds, as a string of characters *h* and *v*.
- **size** (*str*) – Size of the destination pages, as a string that is to be parsed by `papersize.parse_papersize()`. This option is incompatible with *signature* and *folds*.
- **signature** (*tuple[int]*) – Layout of source pages on output pages. For instance `(2, 3)` means: the printed sheets are to be cut in a matrix of 2 horizontal sheets per 3 vertical sheets. This option is incompatible with *size* and *folds*.
- **bind** (*str*) – Binding edge. Can be one of *left*, *right*, *top*, *bottom*.
- **creep** (*function*) – Function that takes the number of sheets in argument, and return the space to be left between two adjacent pages.
- **last** (*int*) – Number of last pages (of the source files) to keep at the end of the output document. If blank pages were to be added to the source files, they would be added before those last pages.
- **group** (*int*) – Group sheets before folding them. See help of command line –group option for more information.

2.2.8 pdfimpose.schema.wire

Cut as invidual pages, stack and wire bind.

Use this schema if you want to print several source pages on each destination page, and your booklet is to be wire-bound.

To use this schema:

- print your imposed PDF file, two-sided;
- cut the sheets to separate the pages (you must get one page per page);
- stack the resulting stacks onto each other;
- bind.

Example:
`source`, `destination`.

WireImpositor

`class pdfimpose.schema.wire.`**`WireImpositor`**(*last: int = 0*, *omargin: ~pdfimpose.schema.Margins | str | ~numbers.Real | ~decimal.Decimal = <factory>*, *mark: list[str] = <factory>*, *imargin: float = 0*, *signature: tuple[int] = (0, 0)*, *back: str = ''*)

Perform imposition of source files, with the 'wire' schema.

impose()

`pdfimpose.schema.wire.`**`impose`**(*files*, *output*, ***, *imargin=0*, *omargin=0*, *last=0*, *mark=None*, *signature=None*, *size=None*)

Perform imposition of source files into an output file, to be cut and "wire bound".

Parameters

- **`files`** (*`list[str]`*) – List of source files (as strings or `io.BytesIO` streams). If empty, reads from standard input.
- **`output`** (*`str`*) – List of output file.
- **`omargin`** (*`float`*) – Output margin, in pt. Can also be a `Margins` object.
- **`imargin`** (*`float`*) – Input margin, in pt.
- **`last`** (*`int`*) – Number of last pages (of the source files) to keep at the end of the output document. If blank pages were to be added to the source files, they would be added before those last pages.
- **`mark`** (*`list[str]`*) – List of marks to add. Only crop marks are supported (*mark=['crop']*); everything else is silently ignored.
- **`signature`** (*`tuple[int]`*) – Layout of source pages on output pages. For instance, `(2, 3)` means that each output page will contain 2 columns and 3 rows of source pages. Incompatible with option *size*.
- **`size`** (*`str | tuple[float]`*) – Size of the output page. Signature is computed to fit the page. This option is incompatible with *signature*.

Deprecated since version 2.5.0: A `perfect` imposition schema did exist before version 2.5.0, when it has been renamed to `hardcover`. It can still be used for backward compatibility, but will be removed in some later version.

CHAPTER THREE

EXAMPLES

- `2024 calendar` (`source`, see LaTeX source file in sources repository).
- Imposition schemas (here are quick examples, more explanation can be found in *Library*):
  - cards: `examples/cards-impose.pdf` (`source`);
  - copycutfold: `examples/copycutfold-impose.pdf` (`source`);
  - cutstackfold: `examples/cutstackfold-impose.pdf` (`source`);
  - onepagezine: `examples/onepagezine-impose.pdf` (`source`);
  - hardcover: `examples/hardcover-impose.pdf` (`source`);
  - saddle: `examples/saddle-impose.pdf` (`source`);
  - wire: `examples/wire-impose.pdf` (`source`).

CHAPTER FOUR

SEE ALSO

I am far from being the first person to implement such an algorithm. I am fond of everything about pre-computer-era printing (roughly, from Gutemberg to the Linotype). Being also a geek, I wondered how to compute how the pages would be arranged on the printer's sheet, and here is the result.

Some (free) other implementation of imposition are:

- Scribus have a list of some of those tools
- BookletImposer
- Impose
- PDF::Imposition (Perl module; I got the idea for some of the schemas from here)

What might make this software better than other is:

- it can perform on arbitrary paper size;
- it can perform several different imposition schemas, without any assumption on folds number.

CHAPTER

FIVE

DOWNLOAD AND INSTALL

See the main project page for instructions, and changelog.

CHAPTER SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

pdfimpose.schema, 6
pdfimpose.schema.cards, 7
pdfimpose.schema.copycutfold, 8
pdfimpose.schema.cutstackfold, 10
pdfimpose.schema.hardcover, 12
pdfimpose.schema.onepagezine, 11
pdfimpose.schema.saddle, 13
pdfimpose.schema.wire, 14

INDEX

A

AbstractImpositor (*class in pdfimpose.schema*), 7

C

CardsImpositor (*class in pdfimpose.schema.cards*), 8
CopyCutFoldImpositor (*class in pdfimpose.schema.copycutfold*), 9
CutStackFoldImpositor (*class in pdfimpose.schema.cutstackfold*), 10

H

HardcoverImpositor (*class in pdfimpose.schema.hardcover*), 12

I

impose() (*in module pdfimpose.schema.cards*), 8
impose() (*in module pdfimpose.schema.copycutfold*), 9
impose() (*in module pdfimpose.schema.cutstackfold*), 10
impose() (*in module pdfimpose.schema.hardcover*), 12
impose() (*in module pdfimpose.schema.onepagezine*), 11
impose() (*in module pdfimpose.schema.saddle*), 14
impose() (*in module pdfimpose.schema.wire*), 15

M

Margins (*class in pdfimpose.schema*), 7
Matrix (*class in pdfimpose.schema*), 7
module
    pdfimpose.schema, 6
    pdfimpose.schema.cards, 7
    pdfimpose.schema.copycutfold, 8
    pdfimpose.schema.cutstackfold, 10
    pdfimpose.schema.hardcover, 12
    pdfimpose.schema.onepagezine, 11
    pdfimpose.schema.saddle, 13
    pdfimpose.schema.wire, 14

O

OnePageZineImpositor (*class in pdfimpose.schema.onepagezine*), 11

P

Page (*class in pdfimpose.schema*), 7
pdfimpose.schema
    module, 6
pdfimpose.schema.cards
    module, 7
pdfimpose.schema.copycutfold
    module, 8
pdfimpose.schema.cutstackfold
    module, 10
pdfimpose.schema.hardcover
    module, 12
pdfimpose.schema.onepagezine
    module, 11
pdfimpose.schema.saddle
    module, 13
pdfimpose.schema.wire
    module, 14

S

SaddleImpositor (*class in pdfimpose.schema.saddle*), 13

W

WireImpositor (*class in pdfimpose.schema.wire*), 15