PyWeaving Documentation

Release 0.0.8.dev

Scott Torborg

March 10, 2016

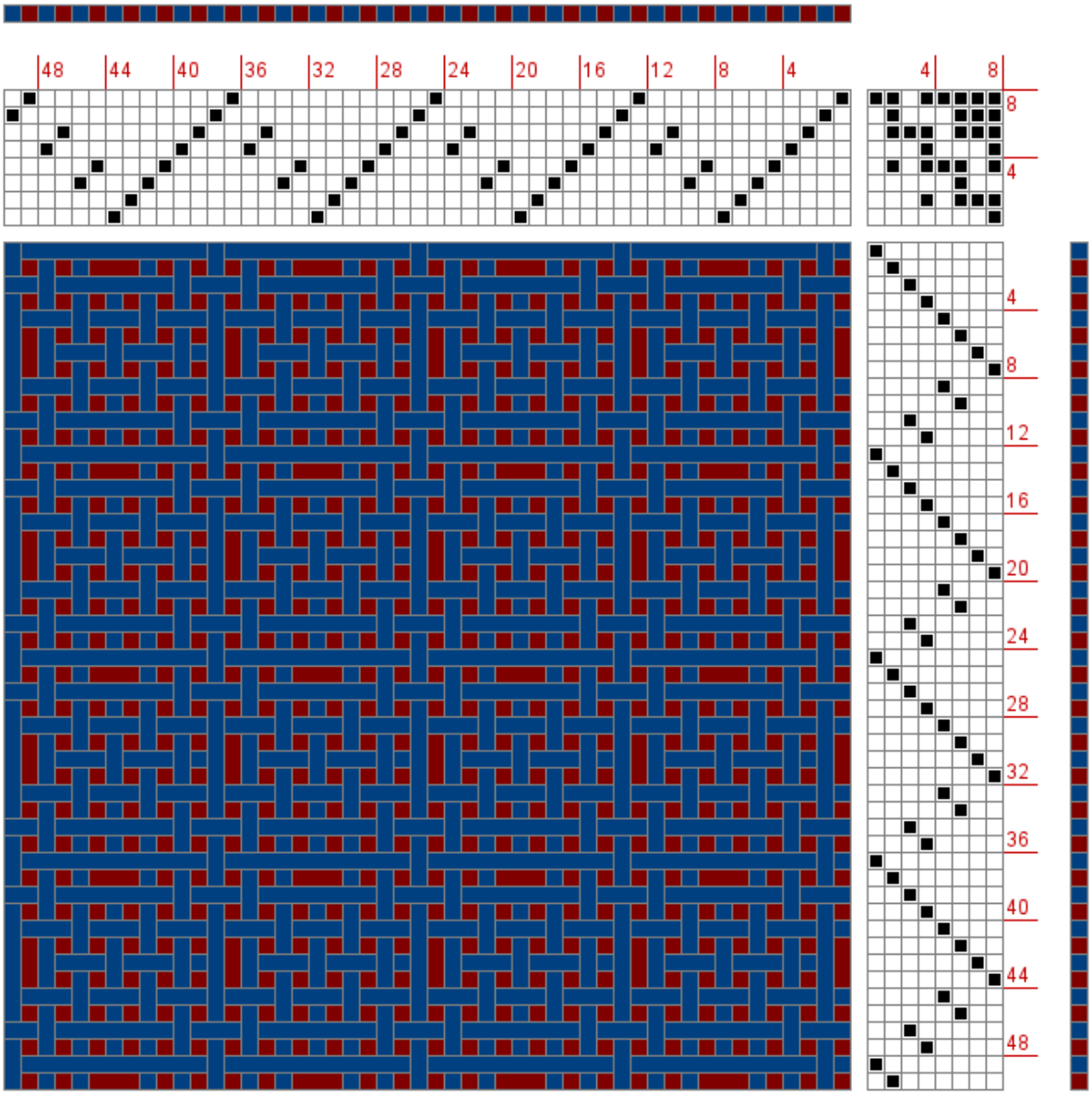
Contents

Author: Scott Torborg

The goal of `pyweaving` is to provide functionality to:

- Parse and write WIF files, with an object-based interface
- Render weave drafts to images
- Render weave drafts to PDF or SVG
- Calculate resource utilization of a pattern (in the form of thread/yarn by color or type)
- Simplify drafts (reduce the number of harnesses or treadles to the bare minimum that are required to produce the same weave, or reduce the number of treadles active at any given time)
- Check a WIF file for validity
- Generate a draft from scratch based on some parameters: for example, simple twills or double weaves
- Interchange with a JSON-based representation of the draft for use in client-side web apps.
- Support rendering drafts with a representation of variable-thickness threads/yarns. E.g. if a given weft thread is narrower than others, that pick should be shorter (fewer pixels tall) than others.

Right now, it provides basic functionality for parsing a WIF file to render a draft. Here's an example:



Draft by Nathaniel Stimson, from HandWeaving.net.

CHAPTER 1

Contents

1.1 Quick Start

1.1.1 Install

The recommended installation method is pip.:

```
$ pip install pyweaving
```

You can also download a source package here directly from the Python Package Index.

PyWeaving can be used via the command line or Python code. Generating a new draft from scratch will generally require writing Python code.

1.1.2 Open a Draft

Lots of great drafts are available in WIF format. A huge repository is Handweaving.net. To open a .wif file as an image, do:

```
$ pyweaving render draft.wif
```

To open a draft from Python, do:

```
from pyweaving import WIFReader

draft = WIFReader('draft.wif').read()
```

1.2 Command Line Usage

PyWeaving includes a command-line utility that can be used for basic tasks. Here are some examples. You can also see the utility itself for more info:

```
$ pyweaving -h
```

1.2.1 Draft Rendering

Render a draft from a WIF file:

```
$ pyweaving render example.wif
```

Or from a JSON file:

```
$ pyweaving render example.json
```

Render a draft to an image:

```
$ pyweaving render example.wif out.png
```

Render a draft to an SVG (vector) file:

```
$ pyweaving render example.wif out.svg
```

Instead of treadling, convert to a liftplan:

```
$ pyweaving render example.wif out.png --liftplan
```

1.2.2 File Conversion

Convert between WIF and JSON:

```
$ pyweaving convert example.wif example.json
```

1.2.3 Instructions

These instructions are interactive, and intend to walk you step-by-step through various processes, providing useful statistics and progress saving along the way.

Show instructions for threading a draft:

```
$ pyweaving thread example.wif
```

Show instructions for weaving:

```
$ pyweaving weave example.wif --liftplan --repeats 50
```

1.3 Tutorial: Creating a Draft

In this tutorial we're going to cover the basic data structures and core APIs of `pyweaving`, and use them to synthesize a somewhat interesting draft from scratch.

WIP

1.4 API Reference

1.4.1 Core Data Model

class `pyweaving.`**`Color`**(*rgb*)

A color type. Internally stored as RGB, and does not support transparency.

`css`

class pyweaving.**Draft**(*num_shafts*, *num_treadles=0*, *liftplan=False*, *rising_shed=True*, *start_at_lowest_thread=True*, *date=None*, *title=u''*, *author=u''*, *address=u''*, *email=u''*, *telephone=u''*, *fax=u''*, *notes=u''*)

The core representation of a weaving draft.

add_warp_thread(*color=None*, *index=None*, *shaft=0*)

Add a warp thread to this draft.

add_weft_thread(*color=None*, *index=None*, *shafts=None*, *treadles=None*)

Add a weft thread to this draft.

advance()

Given a base draft, make it 'advance'. Essentially: 1. Repeat the draft N times, where N is the number of shafts, in both the warp and weft directions. 2. On each successive repeat, offset the threading by 1 additional shaft and the treadling by one additional treadle.

all_threads_attached()

Check whether all threads (weft and warp) will be "attached" to the fabric, instead of just falling off.

compute_drawdown()

Compute a 2D array containing the thread visible at each position.

compute_drawdown_at(*position*)

Return the thread that is on top (visible) at the specified zero-indexed position.

compute_floats()

Return an iterator over every float, yielding a tuple for each one:

```
(start, end, visible, length, thread)
```

FIXME: This ignores the back side of the fabric. Should it?

compute_longest_floats()

Return a tuple indicating the longest floats for warp, weft.

FIXME This might be producing incorrect results.

compute_warp_crossings()

Iterate over each warp row and compute the total number of thread crossings in that row.

compute_weft_crossings()

Iterate over each weft row and compute the total number of thread crossings in that row. Useful for determining sett.

copy()

Make a complete copy of this draft.

flip_warpwise()

Flip/mirror along the warp axis: e.g. looking at the front of the loom, the near side of the fabric becomes the far, and the far becomes the near.

flip_weftwise()

Flip/mirror along the weft axis: e.g. looking at the front of the loom, the left side of the fabric becomes the right, and the right becomes the left.

classmethod from_json(*s*)

Construct a new Draft instance from its JSON representation. Counterpart to `.to_json()`.

invert_shed()

Convert from rising shed to sinking shed, or vice versa. Note that this will actually update the threading/tie-up to preserve the same drawdown: if this is not desired, simply change the .rising_shed attribute.

make_selvedges_continuous(*add_new_shafts=False*)
Make the selvedge threads "continuous": that is, threaded and treadled such that they are picked up on every pick. This method will try to use the liftplan/tieup and switch selvedge threads to alternate shafts. If that is impossible and `add_new_shafts` new shafts will be added to handle the selvedge threads.

FIXME This method works, but it does not necessarily produce the subjectively "best" solution in terms of aesthetics and structure. For example, it may result in longer floats than necessary.

reduce_active_treadles()
Optimize to use the fewest number of active treadles on any given pick, because not every weaver is an octopus. Note that this may mean using more total treadles.

Cannot be called on a liftplan draft.

reduce_shafts()
Optimize to use the fewest number of shafts, to attempt to make a complex draft possible to weave on a loom with fewer shafts. Note that this may make the threading more complex or less periodic.

reduce_treadles()
Optimize to use the fewest number of total treadles, to attempt to make a complex draft possible to weave on a loom with a smaller number of treadles. Note that this may require that more treadles are active on any given pick.

Cannot be called on a liftplan draft.

repeat(*n*)
Given a base draft, make it repeat with N units in each direction.

rotate()
Rotate the draft: the weft becomes the warp, and vice versa.

selvedge_continuous(*low*)
Check whether the selvedge corresponding to the lowest-number thread is continuous.

selvedges_continuous()
Check whether or not both selvedge threads are "continuous" (will be picked up on every pick).

sort_threading()
Reorder the shaft assignment in threading so that it follows as sequential of an order as possible.

For a liftplan draft, will change the threading and liftplan.

For a treadled draft, will change the threading and tieup, won't change the treadling.

sort_treadles()
Reorder the treadle assignment in tieup so that it follows as sequential of an order as possible in treadling.

Will change the tieup and treadling, won't change the threading. If sorting both threading and treadles, call `.sort_threading()` before calling `.sort_treadles()`.

Cannot be called on a liftplan draft.

to_json()
Serialize a Draft to its JSON representation. Counterpart to `.from_json()`.

exception pyweaving.**DraftError**

class pyweaving.**Shaft**
Represents a single shaft of the loom.

class pyweaving.**Treadle**(*shafts=None*)
Represents a single treadle of the loom.

class pyweaving.**WarpThread**(*color=None*, *shaft=None*)
Represents a single warp thread.

class pyweaving.**WeftThread**(*color=None*, *shafts=None*, *treadles=None*)
Represents a single weft thread.

connected_shafts

1.4.2 WIF Import / Export

class pyweaving.wif.**WIFReader**(*filename*)
A reader for a specific WIF file.

allowed_units = (u'decipoints', u'inches', u'centimeters')

getbool(*section*, *option*)

put_metadata(*draft*)

put_tieup(*draft*)

put_warp(*draft*, *wif_palette*)

put_weft(*draft*, *wif_palette*)

read()
Perform the actual parsing, and return a Draft instance.

class pyweaving.wif.**WIFWriter**(*draft*)
A WIF writer for a draft.

write(*filename*, *liftplan=False*)

write_liftplan(*config*)

write_metadata(*config*, *liftplan*)

write_palette(*config*)

write_threading(*config*)

write_threads(*config*, *wif_palette*, *dir*)

write_tieup(*config*)

write_treadling(*config*)

1.4.3 Draft Rendering

class pyweaving.render.**ImageRenderer**(*draft*, *liftplan=None*, *margin_pixels=20*, *scale=10*, *foreground=(127, 127, 127)*, *background=(255, 255, 255)*, *markers=(0, 0, 0)*, *numbering=(200, 0, 0)*)

make_pil_image()

pad_image(*im*)

paint_drawdown(*draw*)

paint_fill_marker(*draw*, *box*)

paint_liftplan(*draw*)

paint_start_indicator(*draw*)

paint_threading(*draw*)

paint_tieup(*draw*)

paint_treadling(*draw*)

paint_warp(*draw*)

paint_weft(*draw*)

save(*filename*)

show()

pyweaving.render.**SVG**

class pyweaving.render.**SVGRenderer**(*draft*, *liftplan=None*, *scale=10*, *foreground=u'#7f7f7f'*, *background=u'#ffffff'*, *markers=u'#000000'*, *numbering=u'#c80000'*)

make_svg_doc()

paint_drawdown(*doc*)

paint_fill_marker(*doc*, *box*)

paint_liftplan(*doc*)

paint_threading(*doc*)

paint_tieup(*doc*)

paint_treadling(*doc*)

paint_warp(*doc*)

paint_weft(*doc*)

render_to_string()

save(*filename*)

write_metadata(*doc*)

class pyweaving.render.**TagGenerator**

1.4.4 Instructions

class pyweaving.instructions.**StatCounter**(*total_picks*, *average_over=10*)

pick()

print_pick_stats()

print_session_stats()

start()

pyweaving.instructions.**describe_interval**(*secs*)
Return a string describing the supplied number of seconds in human-readable time, e.g. "107 hours, 42 minutes".

pyweaving.instructions.**load_save_file**(*save_filename*)

pyweaving.instructions.**print_shafts**(*draft*, *connected*)
Print the shaft lift state, as for a table loom.

pyweaving.instructions.**threading**(*draft*, *repeats=1*, *color_table={0: u'red', 1: u'yellow', 2: u'blue', 3: u'white', 4: u'red', 5: u'yellow', 6: u'blue', 7: u'white', 8: u'red', 9: u'yellow', 10: u'blue', 11: u'white', 12: u'red', 13: u'yellow', 14: u'blue', 15: u'white', 16: u'red', 17: u'yellow', 18: u'blue', 19: u'white', 20: u'red', 21: u'yellow', 22: u'blue', 23: u'white', 24: u'red', 25: u'yellow', 26: u'blue', 27: u'white', 28: u'red', 29: u'yellow', 30: u'blue', 31: u'white', 32: u'red', 33: u'yellow', 34: u'blue', 35: u'white', 36: u'red', 37: u'yellow', 38: u'blue', 39: u'white', 40: u'red', 41: u'yellow', 42: u'blue', 43: u'white', 44: u'red', 45: u'yellow', 46: u'blue', 47: u'white', 48: u'red', 49: u'yellow', 50: u'blue', 51: u'white', 52: u'red', 53: u'yellow', 54: u'blue', 55: u'white', 56: u'red', 57: u'yellow', 58: u'blue', 59: u'white', 60: u'red', 61: u'yellow', 62: u'blue', 63: u'white'}*)

Print threading instructions.

pyweaving.instructions.**tieup**(*draft*)

Print tie-up instructions.

pyweaving.instructions.**wait_for_key**()

pyweaving.instructions.**weaving**(*draft*, *repeats*, *start_repeat*, *start_pick*, *save_filename=None*)

Print weaving instructions. Liftplan only for now.

current_pick, start_repeat, and start_pick are 1-indexed.

pyweaving.instructions.**write_save_file**(*save_filename*, *obj*)

1.4.5 Utility Generators

pyweaving.generators.twill.**twill**(*size=2*, *warp_color=(0, 0, 100)*, *weft_color=(255, 255, 255)*)

pyweaving.generators.tartan.**tartan**(*sett*, *repeats=1*)

pyweaving.generators.dither.**add_rgb**(*a*, *b*)

pyweaving.generators.dither.**closest**(*want_rgb*, **available_rgb*)

pyweaving.generators.dither.**diff_rgb**(*a*, *b*)

pyweaving.generators.dither.**dithered_gradient**(*start_color*, *end_color*, *count*)

Make a dithering sequence which simulates a gradient between two colors across `count` threads. Returns a list of length `count` where each element is the color to be used for the corresponding thread.

pyweaving.generators.dither.**manhattan_distance**(*a*, *b*)

1.5 Contributing

Patches and suggestions are strongly encouraged! GitHub pull requests are preferred, but other mechanisms of feedback are welcome.

A comprehensive test suite is included. To run the tests, simply run in the top level of the repo:

```
$ tox
```

This will also ensure that the Sphinx documentation builds correctly, and that there are no PEP8 or Pyflakes warnings in the codebase.

Any pull requests should preserve all of these things.

See also:

What's WIF Got to Do With It? A brief overview of the history and purpose of WIF.

The WIF Specification The canonical WIF 1.1 specification, published April 1997.

CHAPTER 2

Indices and Tables

- genindex
- modindex

p

A

C

D

F

G

I

L

M

P

R

S

T

W