Spreads

Documentation

Release 0.4

Johannes Baiter

October 13, 2014

Contents

CHAPTER 1

Command-Line Tutorial

This tutorial assumes that you are using a setup with two Canon A2200 cameras that have the latest version of CHDK installed. The rest of the setup is up to you, though development and testing has been performed with a build of the DIYBookScanner (http://diybookscanner.org/forum/viewtopic.php?f=1&t=1192). Furthermore, the following instructions are tailored to an up-to-date installation of a Debian GNU/Linux installation or one of its derivatives (*buntu, Linux Mint, etc.). You might have to adjust the commands for other distributions. This tutorial will also use most of the included plugins, so the dependencies are rather numerous, though you can adapt that, if you want.

The described (and recommended) way to install *spreads* is inside of a virtualenv (http://docs.python-guide.org/en/latest/dev/virtualenvs/), not system-wide, though you can do so as well, if you like.

1.1 Installation

First, ensure that you have all the dependencies installed:

```
$ sudo apt-get install python2.7 python2.7-dev python-virtualenv libusb-dev\
  libjpeg-dev libtiff-dev libqt4-core rubygems ruby-rmagick libmagickwand-dev\
  libhpricot-ruby scantailor
$ sudo gem install pdfbeads
$ wget http://djvubind.googlecode.com/files/djvubind_1.2.1.deb
$ sudo dpkg -i djvubind_1.2.1.deb
# Download the latest 'chdkptp' release from the website:
# https://www.assembla.com/spaces/chdkptp/documents
$ sudo unzip chdkptp-<version>-<platform>.zip -d /usr/local/lib/chdkptp
$ virtualenv ~/.spreads
$ source ~/.spreads/bin/activate
$ pip install spreads
```

1.2 Configuration

To perform the initial configuration, launch the *configure* subcommand:

```
$ spread configure
```

You will be asked to select a device driver (choose **a2200**) and some plugins (choose all except **gui** and **colorcorrect**). Next, configure the order in which your postprocessing plugins should be invoked. I recommend you set this to the following value:

```
autorotate,scantailor,tesseract
```

Next, you can set the target pages for each of your cameras. This is necessary, as the application has to:

- combine the images from both cameras to a single directory in the right order
- set the correct rotation for the captured images

To do both of these things automatically, the application needs to know if the image is showing an odd or even page. Don't worry, you only have to perform this step once, the orientation is stored on the camera's memory card (under *A/OWN.TXT*). Should you later wish to briefly flip the target pages, you can do so via a command-line flag.

Note: If you are using a DIYBookScanner, the device for *odd* pages is the camera on the **left**, the one for *even* pages on the **right**.

After that, you can choose to setup the *focus* for your devices. By default, the focus will be automatically detected on each shot. But this can lead to problems: Since the camera uses the center of the frame to obtain its focus, your images will be out of focus in cases where the center of the page does not have any text on it, e.g. in chapter endings. This step is therefore recommended for most users. Before you continue, make sure that you have loaded a book into the scanner, and that the pages facing the camera are evenly filled with text or illustrations.

Once you're done, you can find the configuration file in the *.config/spreads* folder in your home directory.

See also:

Configuring

1.3 Workflow

To begin, we run *spreads* in the **wizard** mode, which will guide us through the whole workflow:

```
$ spread wizard ~/my_book
```

On startup, your cameras will simultaneously adjust their zoom levels and set their focus. Once this is done, the application will ask you to press one of your configured *shooting keys* (default: **b** or **space**). If you do so, both cameras will take a picture simultaneously, which is then transferred to our computer and stored under the correct filename in the *raw* subdirectory of our project directory. Should you notice that you made a mistake during the last capture, you can press **r** do discard the last capture and retake it. Now scan as many pages as you need, when you're done, press **f** to quit the capturing process and continue to the next step.

Now spreads will begin with the postprocessing of the captured images. If you followed the instructions so far, it will first rotate the images, which, depending on your CPU and the number of images might take a minute or two. Afterwards, *spreads* will launch a **ScanTailor** process in the background, that will generate a configuration file (stored under *~/my_book/my_book.ScanTailor*). When it has finished, it will open the ScanTailor GUI, so you can make your final adjustments to the configuration. Save and close your project when you're finished. *spreads* will then split the configuration file into as many files as your computer has CPU cores and perform the final ScanTailor step on all of them in parallel.

Finally, once ScanTailor has completed generating the final version of your images (in the *done* folder), it will generate PDF and DJVU files from them, which you will find under the *~/my_book* directory.

If you want to know more about any of the above steps and how you can configure them, check out the entries for the appropriate *appropriate plugins*.

CHAPTER 2

GUI Wizard

2.1 Enabling the GUI

To enable the GUI wizard, first make sure that you have an up-to date version of PySide installed on your machine and linked to your virtual environment:

```
$ sudo apt-get install python-pyside
$ ln -s /usr/lib/python2.7/dist-packages/PySide ~/.spreads/lib/python2.7/site-packages/P
```

Then, just re-run the *configure* step and add *gui* to your list of plugins. You can launch the GUI with the following command:

```
$ spread gui
```

2.2 Usage

On the *first screen*, you can adjust various settings for your scan. You have to specify a project directory before you can continue. The rest of the settings depends on which plugins you have enabled. Select the plugin to configure from the dropdown menu and make your adjustments.

*After you've clicked *next**, the cameras will be prepared for capture by setting their zoom and focus levels. At the top of the screen you can see how many pages you've already scanned, as well as your current average scanning speed. The text box at the bottom of the screen will display any warnings or error messages that occur during the capture process. Next, initiate a capture by clicking on the button (or pressing one of the capture keys).

Once you have *captured your first pages*, you will see the last two pages your cameras shot. Here you can verify that everything went as expected. Should you notice a mistake, you can discard the previous shot and retake it by clicking on the *retake* button.

Once you've finished scanning your book and *clicked on the *next* button,* spreads will execute all enabled postprocessing plugins in the sequence that you configured. You can verify the progress in the text box.

Last, spreads will assemble the processed scans into your enabled output formats. As in the postprocessing step, follow the progress via the text box.

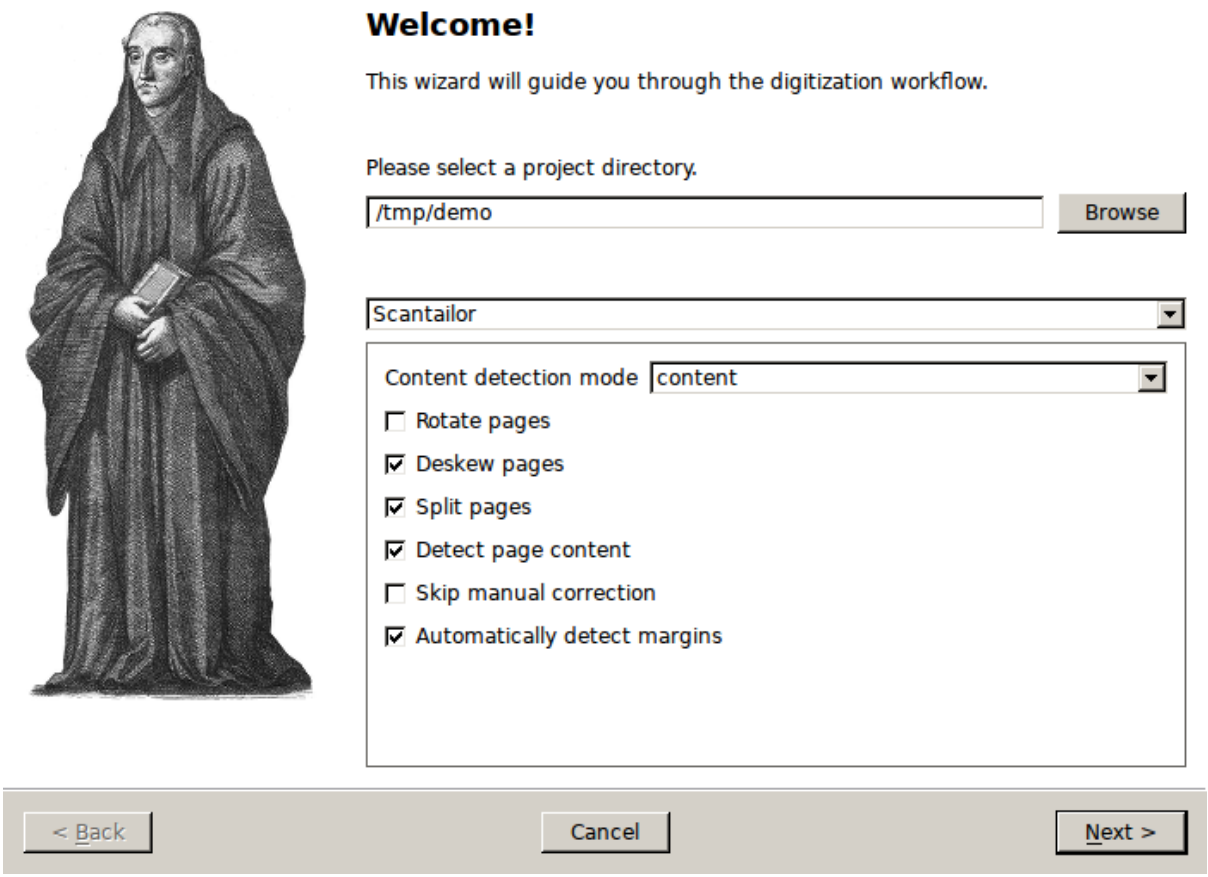

Figure 2.1: Initial setup page

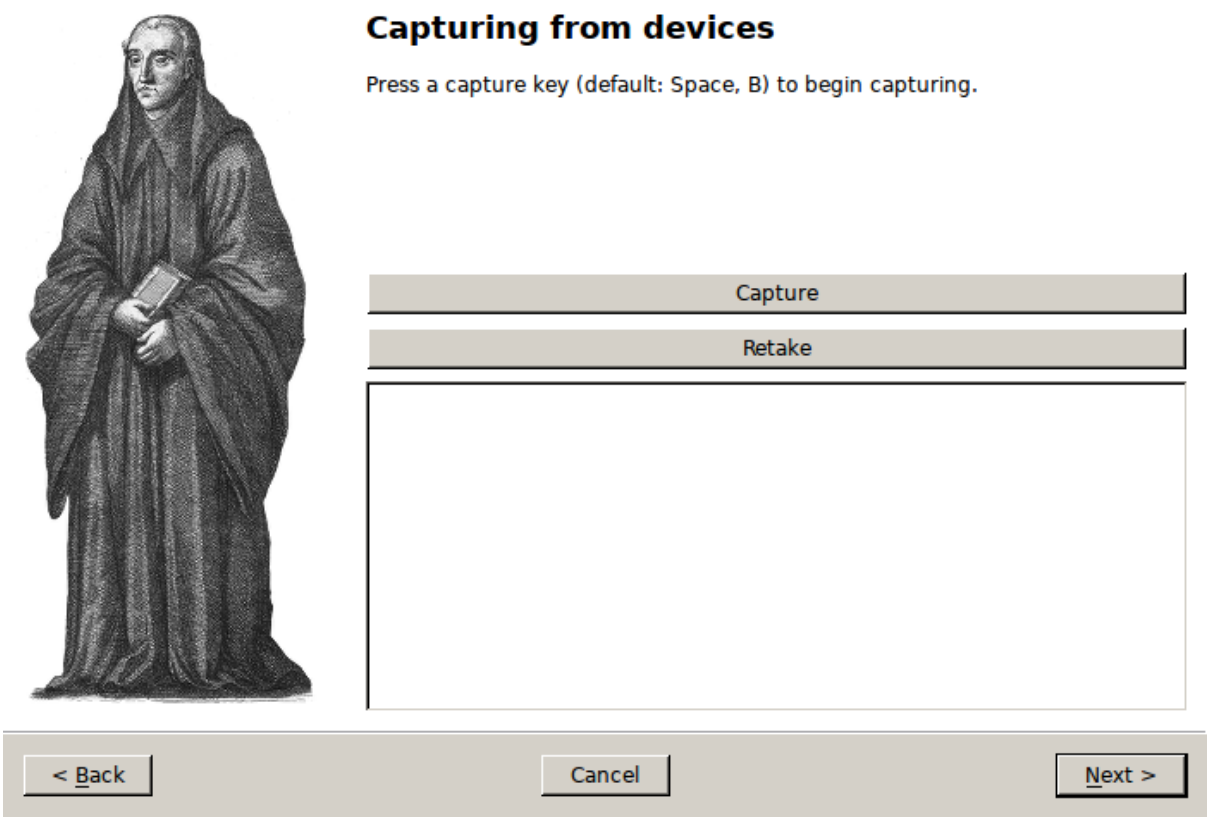

Figure 2.2: Capture page

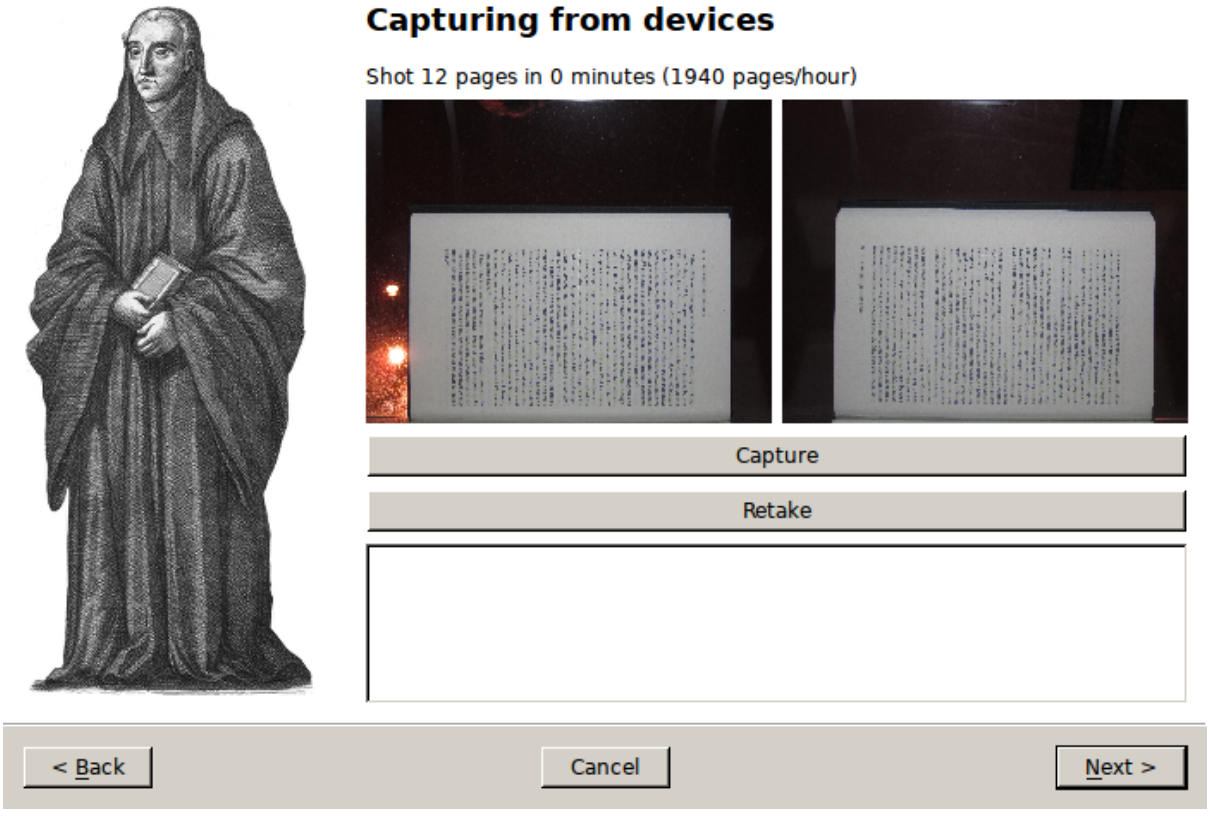

Figure 2.3: Capture page with control images

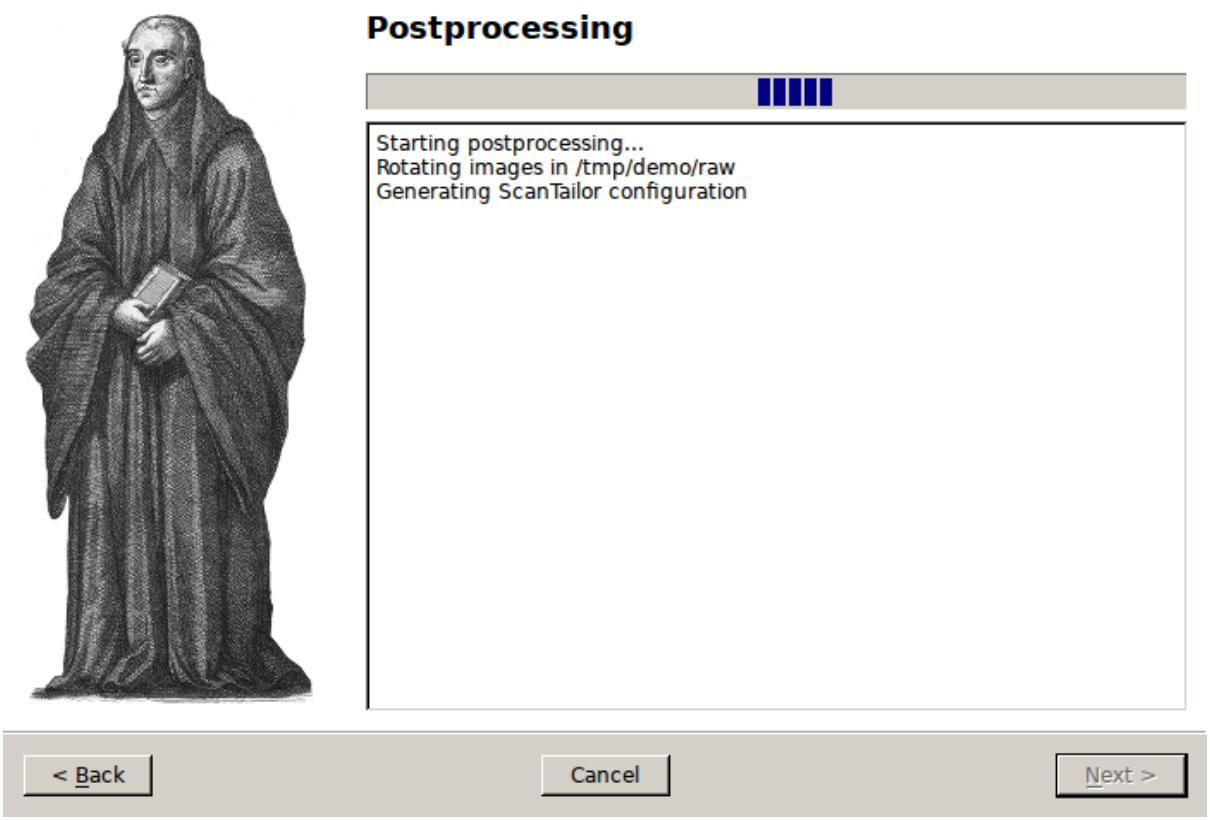

Figure 2.4: Postprocessing page

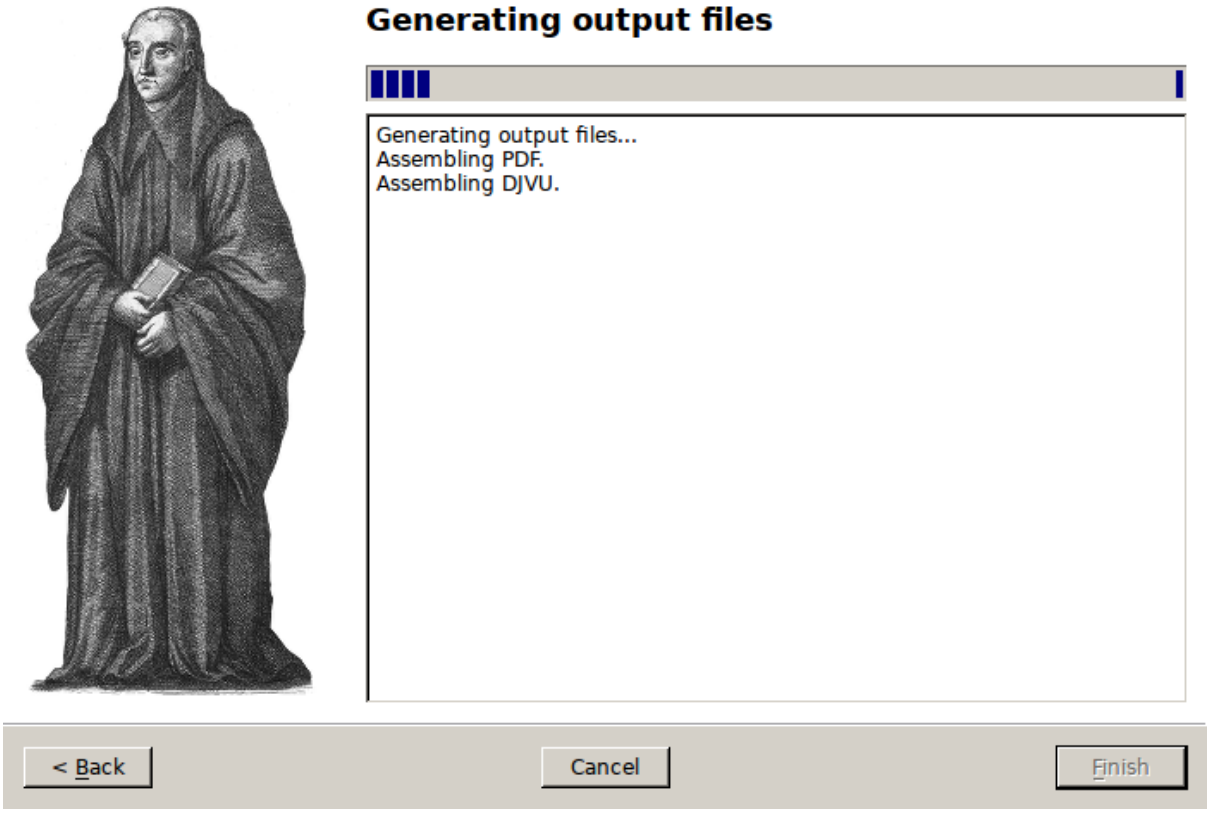

Figure 2.5: Output page

CHAPTER 3

Installation

3.1 Prerequisites

- Python 2.7 with pip (http://www.pip-installer.org) installed

3.2 Optional requirements

To use some of the included plugins, you might want to install the following dependencies:

- chdkptp (https://www.assembla.com/spaces/chdkptp/wiki) to use cameras with the CHDK firmware (installed in */usr/local/lib/chdkptp*)
- An up-to date version of ScanTailor-enhanced (http://sourceforge.net/p/scantailor/code/ci/enhanced/tree/)
- pdfbeads (http://rubygems.org/gems/pdfbeads)
- djvubind (http://code.google.com/p/djvubind/)
- PySide (http://pyside.org) (available as *python-pyside* for Debian and Ubuntu)

3.3 Installing from PyPi

This will grab the latest release and install all Python dependencies:

```
$ sudo pip install spreads
```

3.4 Installing from GitHub

Like from PyPi, only using the development version from GitHub (might break, use with caution!):

```
$ sudo pip install git+git://github.com/jbaiter/spreads.git@master
```

CHAPTER 4

Configuration

Upon first launch, *spreads* writes a configuration file to *~/.config/spreads/config.yaml*. In it, you can change all of the available settings to your liking. The configuration options are the same ones that you can set on the command-line, so just call *spreads <command> –help* to view the documentation.

```
# Valid values: 'none', 'debug', 'info', 'warning', 'error', 'critical'
loglevel: warning
plugins:
    - autorotate
    - scantailor
    - tesseract
    - gui
    - pdfbeads
    - djvubind

# Options for 'capture' step
capture:
     capture_keys: [' ', b]
driver: dummy
colorcorrect:
     true_blue: 119
     true_green: 119
     true_red: 119
device:
     focus_distance: 384
     dpi: 300
     parallel_capture: yes
     chdkptp_path: /usr/local/lib/chdkptp
     zoom_level: 3
     shoot_raw: no
     sensitivity: 80
     flip_target_pages: no
     shutter_speed: 1/25
tesseract:
     language: deu-frak
autorotate:
     rotate_even: 90
     rotate_odd: -90
scantailor:
     margins:
        - 2.5
        - 2.5
        - 2.5
        - 2.5
```

```
auto_margins: yes
autopilot: no
split_pages: yes
deskew: yes
rotate: no
detection: content
content: yes
```

CHAPTER 5

Command-Line Interface

spread is *spreads'* command-line interface.

It takes a *command* as its first argument:

```
$ spread [--verbose] COMMAND [ARGS...]
```

All of *spreads'* functionality is accessible via the following commands:

5.1 wizard

```
$ spread wizard <project-path>
```

Start *spreads* in wizard mode. This will go through all of the steps outlined below and store images and output files in *project-path*

5.2 configure

```
$ spread configure
```

This command lets you select a device driver and a set of plugins to activate. It also allows you to set the target pages for your devices, in case you are using two devices for capturing.

5.3 capture

```
$ spread capture [OPTIONS] <project-director>
```

This command will start a capturing workflow. Make sure that your devices are turned on. After the application is done setting them up, you will enter a loop, where all devices will trigger simultaneously (if not configured otherwise, see below) when you press one of the capture keys (by default: the **b** or **spacebar** key). Press *r* to discard the last capture and retake it. Press *f* to finish the capture process.

--no-parallel-capture
: When using two devices, do not trigger them simultaneously but one after the other.

--flip-target-pages
: When using two devices, flip the configured target pages, i.e. the camera configured to be *odd*

will temporarily be the *even* device and vice versa. This can be useful when you are scanning e.g. East-Asian literature.

5.4 postprocess

```
$ spread postprocess [--jobs <int>] <project-directory>
```

Start the postprocessing workflow by calling each of the *postprocessing plugins* defined in the configuration one after the other.The transformed images will be stored in *project-directory/done*.

--jobs number-of-jobs, **-j** number-of-jobs
: Specify how many concurrent processes should be used for rotation and ScanTailor. By default, *spreads* will use as many as CPU cores are available.

5.5 output

```
$ spread output <project-directory>
```

Start the output workflow, calling each of the *output plugins* defined in the configuration. All output files will be stored in *project-directory/out*.

CHAPTER 6

Plugins

spreads comes with a variety of plugins pre-installed. Plugins perform their actions at several designated points in the workflow. They can also add specify options that can be set from one of the interfaces.

6.1 subcommand plugins

These plugins add additional commands to the *spread* application. This way, plugins can implement additional workflow steps or provide alternative interfaces for the application.

6.1.1 gui

Launches a graphical interface to the workflow. The steps are the same as with the *CLI wizard*, additionally a small thumbnail of every captured image is shown during the capture process. Requires an installation of the *PySide* packages. Refer to the *GUI tutorial* for more information.

6.2 *postprocess* plugins

An extension to the *postprocess* command. Performs one or more actions that either modify the captured images or generate a different output.

6.2.1 autorotate

Automatically rotates the images according to their device of origin. By default this means -90° for odd pages and 90° for even pages, but these can be set to arbitrary values by specifying the `rotate-even` or `rotate-odd` options. You probably want to stick to multiples of 90°.

`--rotate-even`
: Change rotation for images from even book pages (default: 90°)

`--rotate-odd`
: See above, only for odd pages (default: -90°)

6.2.2 colorcorrect

Automatically fixes white balance for your scanned images. To use it, enable it in the configuration, set the RGB values for your grey cards and ensure that the first two images you take are of your grey cards.

6.2.3 scantailor

Automatically generate a ScanTailor configuration file for your scanned book and generate output images from it. After the configuration has been generated, you can adjust it in the ScanTailor UI, that will be opened automatically, unless you specified the `auto` option. The generation of the output images will run on all CPU cores in parallel.

--autopilot
Run ScanTailor on on autopilot and do not require and user input during postprocessing. This skips the step where you can manually adjust the ScanTailor configuration.

--detection `<content/page> [default: content]`
By default, ScanTailor will use content boundaries to determine what to include in its output. With this option, you can tell it to use the page boundaries instead.

--no-content
Disable content detection step.

--rotate
Enable rotation step.

--no-deskew
Do not deskew images.

--no-split-pages
Do not split pages.

--no-auto-margins
Disable automatically detect margins.

6.2.4 tesseract

Perform optical character recognition on the scanned pages, using the *tesseract* application, that has to be installed in order for the plugin to work. For every recognized page, a HTML document in hOCR format will be written to *project-directory/done*. These files can be used by the output plugins to include the recognized text.

--language `LANGUAGE`
Tell tesseract which language to use for OCR. You can get a list of all installed languages on your system by running *spread capture –help*.

6.3 *output* plugins

An extension to the *out* command. Generates one or more output files from the scanned and postprocessed images. Writes its output to *project-directory/done*.

6.3.1 pdfbeads

Generate a PDF file from the scanned and postprocessed images, using the *pdfbeads* tool. If OCR has been performed before, the PDF will include a hidden text layer with the recognized text.

6.3.2 djvubind

Generate a DJVU file from the scanned and postprocessed images, using the *djvubind* tool.

See also:

Extending spreads functionality

CHAPTER 7

Extending *spreads*

7.1 Adding support for new devices

To support new devices, you have to subclass `DevicePlugin` in your module and add it as an entry point for the `spreadsplug.devices` namespace to your package's `setup.py`. In it, you override and implement the features supported by your device. Take a look at the plugin for CHDK-based cameras (https://github.com/jbaiter/spreads/blob/master/spreadsplug/dev/chdkcamera.py) and the relevant part of spreads' setup.py (https://github.com/jbaiter/spreads/blob/master/setup.py) for a reference implementation.

Devices are assigned a `DevicePlugin` implementation based on their USB device's properties. This means that you can support a whole range of devices with a single `DevicePlugin` implementation, if you know a set of attributes that apply to all of them.

7.2 Extending *spreads* built-in commands

You can extend all of *spread's* built-in commands with your own code. To do, you just have to inherit from the `HookPlugin` class and implement one or more of its hooks. Furthermore, you have to add an entry point for that class in the `spreadsplug.hooks` namespace in your package's `setup.py` file. For a list of available hooks and their options, refer to the *API documentation*. Example implementations can be found on GitHub (https://github.com/jbaiter/spreads/blob/master/spreadsplug)

See also:

module `spreads.plugin`, module `spreads.util`

7.3 Adding new commands

You can also add entirely new commands to the application. Simply subclass `HookPlugin` again, implement the `add_command_parser` method and add your new class as an entry point to the `spreadsplug.hooks` namespace. Your plugin class will most probably be a very few lines, telling the CLI parser its name, arguments and pass a function that will do the main work.

CHAPTER 8

Frequently Asked Questions

8.1 CHDK Cameras

... When capturing, the commands frequently time out.

> This is a known issue when both cameras are connected to the same USB hub. It seems to occur less frequently with powered USB hubs, but the safest way to avoid these hickups is to connect each device to a separate USB hub/port. You might also want to try another USB cable.

... `USBError: [Errno 13] Access denied (insufficient permissions)`

> This means that your user is not allowed to write to the camera devices. To temporarily fix this, run `$ sudo chmod -R a+rw /dev/bus/usb/*`. To permanently fix the permissions, create a new udev rule that sets the permissions when the devices are plugged in.

CHAPTER 9

API Reference

9.1 spreads.plugin

class spreads.plugin.**PluginOption**(*value*, *docstring=None*, *selectable=False*)
A configuration option.

Attr value The default value for the option or a list of available options if :attr selectable: is True

Attr docstring A string explaining the configuration option

Attr selectable Make the PluginOption a selectable, i.e. value contains a list or tuple of acceptable values for this option, with the first member being the default selection.

__init__(*value*, *docstring=None*, *selectable=False*)

class spreads.plugin.**SpreadsPlugin**(*config*)
Plugin base class.

classmethod configuration_template()
Allows a plugin to define its configuration keys.

The returned dictionary has to be flat (i.e. no nested dicts) and contain a PluginOption object for each key.

Example:

```
{
 'a_setting': PluginOption(value='default_value'),
 'another_setting': PluginOption(value=[1, 2, 3],
                                 docstring="A list of things"),
 # In this case, 'full-fat' would be the default value
 'milk': PluginOption(value=('full-fat', 'skim'),
                      docstring="Type of milk",
                      selectable=True),
}
```

Returns dict with unicode: PluginOption(value, docstring, selection)

__init__(*config*)
Initialize the plugin.

Parameters **config** (*confit.ConfigView*) – The global configuration object, by default only the section with plugin-specific values gets stored in the *config* attribute, if the plugin has a *__name__* attribute.

class spreads.plugin.**DevicePlugin**(*config*, *device*)

Base class for devices.

Subclass to implement support for different devices.

features = ()

Tuple of DeviceFeatures constants that designate the features the device offers.

__init__(*config*, *device*)

Set connection information and other properties.

Parameters

- **config** (*spreads.confit.ConfigView*) – spreads configuration
- **device** (usb.core.Device (http://github.com/walac/pyusb)) – USB device to use for the object

set_target_page(*target_page*)

Set the device target page, if applicable.

Parameters **target_page** (*unicode in (u"odd", u"even")*) – The target page

prepare_capture(*path*)

Prepare device for scanning.

What this means exactly is up to the implementation and the type, of device, usually it involves things like switching into record mode, path and applying all relevant settings.

Parameters **path** (*pathlib.Path*) – Project base path

capture(*path*)

Capture a single image with the device.

Parameters **path** (*pathlib.Path*) – Path for the image

class spreads.plugin.**HookPlugin**(*config*)

Add functionality to any of spreads' commands by implementing one or more of the available hooks.

classmethod add_command_parser(*rootparser*)

Allows a plugin to register a new command with the command-line parser. The subparser that is added to :param rootparser: should set the class' __call__ method as the func (via set_defaults) that is executed when the subcommand is specified on the CLI.

Parameters **rootparser** (*argparse.ArgumentParser*) – The root parser that this plugin should add a subparser to.

prepare_capture(*devices*, *path*)

Perform some action before capturing begins.

Parameters

- **devices** (*list(DevicePlugin)*) – The devices used for capturing

- **path** (*pathlib.Path*) – Project path

capture (*devices*, *path*)
Perform some action after each successful capture.

Parameters

- **devices** (*list(DevicePlugin)*) – The devices used for capturing
- **path** (*pathlib.Path*) – Project path

finish_capture (*devices*, *path*)
Perform some action after capturing has finished.

Parameters

- **devices** (*list(DevicePlugin)*) – The devices used for capturing
- **path** (*pathlib.Path*) – Project path

process (*path*)

Perform one or more actions that either modify the captured images or generate a different output.

Parameters **path** (*pathlib.Path*) – Project path

output (*path*)
Assemble an output file from the postprocessed images.

Parameters **path** (*pathlib.Path*) – Project path

__init__ (*config*)
Initialize the plugin.

Parameters **config** (*confit.ConfigView*) – The global configuration object, by default only the section with plugin-specific values gets stored in the *config* attribute, if the plugin has a *__name__* attribute.

classmethod configuration_template ()
Allows a plugin to define its configuration keys.

The returned dictionary has to be flat (i.e. no nested dicts) and contain a PluginOption object for each key.

Example:

```
{
 'a_setting': PluginOption(value='default_value'),
 'another_setting': PluginOption(value=[1, 2, 3],
                                 docstring="A list of things"),
 # In this case, 'full-fat' would be the default value
 'milk': PluginOption(value=('full-fat', 'skim'),
                      docstring="Type of milk",
                      selectable=True),
}
```

Returns dict with unicode: PluginOption(value, docstring, selection)

spreads.plugin.**get_devices**(*config*)
Initialize configured devices.

spreads.plugin.**get_relevant_extensions**(*plugin_manager*, *hooks*)
Find all extensions that implement certain hooks.

Parameters **hooks** (*list(unicode)*) – A list of hook method names

Returns A generator that yields relevant extensions

Return type generator(Extension)

9.2 spreads.util

Various utility functions.

spreads.util.**find_in_path**(*name*)
Find executable in $PATH.

Parameters **name** (*unicode*) – name of the executable

Returns bool – True if *name* is found or False

class spreads.util.**abstractclassmethod**(*func*)
New decorator class that implements the @abstractclassmethod decorator added in Python 3.3 for Python 2.7.

Kudos to http://stackoverflow.com/a/13640018/487903

__init__(*func*)

class spreads.util.**ColourStreamHandler**(*stream=None*)
A colorized output SteamHandler Kudos to Leigh MacDonald: http://leigh.cudd.li/article/Cross_Platform_Colorized_Logger_Output_Using_Pythons_logging_Module_And_Co

is_tty
Check if we are using a "real" TTY. If we are not using a TTY it means that the colour output should be disabled.

Returns Using a TTY status

Return type bool

CHAPTER 10

Changelog

10.1 0.4.2 (2014/01/05)

- Fix packaging issues
- Small bugfix for older Tesseract versions

10.2 0.4.1 (2013/12/25)

- Fix 'spread' tool
- Include missing *vendor* package in distribution

10.3 0.4 (2013/12/25)

- Use *chdkptp* utility for controlling cameras with CHDK firmware
- Fix instability when shooting with CHDK cameras
- Shoot images in RAW/DNG file format *(experimental)*
- Remove *download* step, images will be directly streamed to the project directory
- Remove *combine* plugin, images will be combined in *capture* step
- Device driver and plugins, as well as their order of execution can be set interactively via the *configure* subcommand, which has to be run before the first usage.
- Lots of internal API changes

10.4 0.3.3 (2013/08/28)

- Fix typo in device manager that prevent drivers from being loaded

10.5 0.3.2 (2013/08/24)

- Fixes a critical bug in the devices drivers

10.6 0.3.1 (2013/08/23)

- Fixes a bug that prevented spreads to be installed

10.7 0.3 (2013/08/23)

- Plugins can add completely new subcommands.
- GUI plugin that provides a graphical workflow wizard.
- Tesseract plugin that can perform OCR on captured images.
- pdfbeads plugin can include recognized text in a hidden layer if OCR has been performed beforehand.
- Use EXIF tags to persist orientation information instead of JPEG comments.
- Better logging with colorized output
- Simplified multithreading/multiprocessing code
- CHDK driver is a lot more stable now

10.8 0.2 (2013/06/30)

- New plugin system based on Doug Hellmann's *stevedore* package, allows packages to extend spreads without being included in the core distribution
- The driver for CHDK cameras no longer relies on gphoto2 and ptpcam, but relies on Abel Deuring's *pyptpchdk* package to communicate with the cameras.
- *Wand* is now used to deal with image data instead of *Pillow*
- New 'colorcorrection' plugin allows users to automatically correct white balance.
- Improved tutorial

10.9 0.1 (2013/06/23)

- Initial release

S

Symbols

A

C

D

F

G

H

I

O

P

S