Spreads

Documentation

Release 0.5git20150526.c802

Johannes Baiter

May 26, 2015

Contents

CHAPTER 1

Installation

1.1 Prerequisites

- Python 2.7 with a recent version of pip[1] installed

1.2 Install requirements

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

- chdkptp[2] to use cameras with the CHDK firmware (installed in */usr/local/lib/chdkptp*)
- An up-to date version of ScanTailor-enhanced[3]
- pdfbeads[4]
- djvubind[5]
- PySide[6] (available as *python-pyside* for Debian and Ubuntu)
- libgphoto2[7]

1.3 Installing the core from PyPi

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

```
$ sudo pip install spreads
```

1.4 Installing plugin dependencies

This will grab all Python dependencies for the selected plugins:

[1]http://www.pip-installer.org
[2]https://www.assembla.com/spaces/chdkptp/wiki
[3]http://sourceforge.net/p/scantailor/code/ci/enhanced/tree/
[4]http://rubygems.org/gems/pdfbeads
[5]http://code.google.com/p/djvubind/
[6]http://pyside.org
[7]http://www.gphoto.org

```
$ sudo pip install spreads[chdkcamera,web,hidtrigger]
```

Adjust the list of plugins as needed.

1.5 Installing a nightly build

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

```
$ sudo pip install http://buildbot.diybookscanner.org/nightly/spreads-latest.tar.gz
```

CHAPTER 2

Configuration

2.1 Initial configuration

To perform the initial configuration, launch the either the *configure* subcommand or its graphical counterpart, *guiconfigure*:

```
$ spread configure
# or
$ spread guiconfigure
```

The following instructions are mostly target at users of the CLI configuration interface, but all of the available settings are also equally available from the GUI and should be pretty self-explanatory.

You will be asked to select a device driver and some plugins. Next, configure the order in which your postprocessing plugins should be invoked. Think of it as a pipelining system, where each of the plugin gets fed the output of its predecessor.

Next, if you are using two cameras for scanning, your can 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 the *–flip-target-pages* command-line flag.

Note: If you are using a DIYBookScanner and the book is facing you, 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.

2.2 Configuration file

spreads writes its 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. Command-line flags that begin with *–no-...* should be entered without the *no* prefix and have *yes* or *no* as their value.

Here is an example that demonstrates the general layout:

```
# Names of activated plugins, postprocessing plugins will be called
# in the order that they are entered here
plugins: [gui, autorotate, scantailor]

# Name of the device driver
driver: chdkcamera

core:
    # Enable verbose output on command-line
    verbose: no
    # Keys that trigger a capture in command-line interface
    capture_keys: [' ', b]
    # Path to logfile
    logfile: ~/.config/spreads/spreads.log
    # Loglevel for logfile
    loglevel: info

# Device settings
device:
    parallel_capture: yes
    flip_target_pages: no

# Plugin settings
tesseract:
    language: deu-frak

scantailor:
    autopilot: no
```

CHAPTER 3

SpreadPi Setup

Materials needed:

- Raspberry Pi (Model B+ recommended)
- network cable
- Class10 SD Card (lower clases will slow down operations *significantly*). See this list[1] for SD cards known to work well with the Raspberry Pi.
- free ethernet port in your router/switch

1. Download the latest version of the SpreadPi disk image of SpreadPi from the buildbot[2]. It contains a fully configured Linux operating system and a complete installation of Spreads, ready to run.
2. Extract the image with 7-Zip[3] and follow the tutorial matching your operating system to copy SpreadPi to the SD-Card that goes into the Raspberry Pi: Windows[4] / OS X[5] / Linux[6].

Note: For most situations, this is all you need to configure the Pi. For advanced users and occasional problematic setups, it is possible to SSH into the Pi and configure it manually. You have to use the following credentials:

Username: *spreads*

Password *spreads*

Root-Password: *raspberry*

3. Now that the Pi has an operating system, we need to configure our devices. SpreadPi currently assumes that the user is running CHDK devices, so check the driver documentation for how to correctly set up the cameras.

5. Connect the network cable to the Pi and your router or switch. Connect all devices. Turn on the devices first, and only then turn on the Pi. The Pi takes a few minutes to boot for the first time - be patient. It will reboot once to resize the image to fit the whole SD-Card.

[1] http://elinux.org/RPi_SD_cards#SD_card_performance
[2] http://buildbot.diybookscanner.org/nightly
[3] http://www.7-zip.org/download.html
[4] http://elinux.org/RPi_Easy_SD_Card_Setup#Flashing_the_SD_Card_using_Windows
[5] http://elinux.org/RPi_Easy_SD_Card_Setup#Flashing_the_SD_card_using_Mac_OSX
[6] http://elinux.org/RPi_Easy_SD_Card_Setup#Flashing_the_SD_Card_using_Linux_.28including_on_a_Pi.21.29

Spreads is getting an IP address from your network and will display that IP address on the screens of your cameras for you when it is ready to begin.

6. Spreads has an easy-to-use web interface. Open a browser on any device that is on the same network as your scanner. If your smartphone or tablet is on your home WiFi network, you can use it to the scanner. To connect to it, enter the IP address that was displayed on the camera screen. Refer to the web plugin documentation for more information on how to use the interface.

CHAPTER 4

Web Interface

4.1 Installation

To install the required dependencies for the web plugin, run the following command:

```
$ pip install spreads[web]
```

Alternatively, make sure you have the following modules installed in their most recent versions:

- Flask
- Flask-Compress
- jpegtran-cffi
- requests
- waitress
- zipstream

To use the JavaScript web interface, make sure you use a recent version of Firefox or Chrome.

4.2 Startup and Configuration

You can launch the web interface with its subcommand:

```
$ spread web [OPTIONS]
```

This will serve the spreads web interface and its RESTish-API for the whole network. There are a number of options available:

--port `<int>`
: Port that the web application is listening on. By default this is *5000*

--mode `<full/scanner/processor>`
: Mode to run the web plugin in. *scanner* only exposes functionality that is needed for scanning, while *processor* only exposes functionality that is needed for postprocessing and output generation. *full* exposes all available functionality. Instances of spreads running in *scanner* mode can transfer their workflows to other instances on the network that run in *processor* mode and let them take care of the postprocessing and output generation.

`--postprocessing-server` `<address>`
Select a default postprocesisng server to user. This is only useful if the web plugin is running in *scanner* mode and the user is planning to transfer workflows to another spreads instance on the network (see above). This configures a default address for such a server that is always shown.

`--standalone-device`
Enable standalone mode. This option can be used for devices that are dedicated to scanning (e.g. a RaspberryPi that runs spreads and nothing else). At the moment the only additional features it enables is the ability to shutdown and reboot the device from the web interface and REST API.

`--debug`
Run the application in debugging mode. This activates source maps in the client-side code, which will increase the initial loading time significantly.

`--project-dir` `<path>`
Location where workflow files are stored. By default this is *~/scans*.

4.3 Interface

You can connect to the interface by opening your browser on an address that looks like this:

```
http://<host-ip-address>:<web-port>
```

If you are running spreads in your local machine, using *localhost* or *127.0.0.1* for the IP address will be enough. If you are running it on a remote machine, you will have to find out its IP address. When you are using CHDK cameras and have them turned on when you launch spreads, their displays will show the IP address of the computer they are connected to. The *web-port* is by default configured to be *5000*, though this can be configured.

The **initial screen** will list all previously created workflows with a small preview image and some information on their status. On clicking one of the workflows, you will be taken to its details page where you can view all of the images and see more information on it. You can also choose to download a ZIP or TAR file with the workflow, containing all images and a configuration file.

From the navigation bar, you can choose to **create a new workflow**. The only metadata you absolutely have to enter is the workflow name. Note that when you enter a name, you will be offered a selection of ISBN records that might match your title. If you select one of these, the rest of the fields will be filled out automatically.

You can also change driver and plugin settings for this workflow by selecting either one from the dropdown menu. For a reference on what the various options mean, please consult the documentation of the repsective plugin or driver. When you are done, you can submit the workflow and the application will take you to the capture screen.

On the **capture screen**, you can see two small review images with which you can verify that the last capture went well. Trigger a new capture by clicking the appropriate button and you will see the images update.

If you spotted an error, you can click the *Retake* button, which will discard the last capture and trigger a new one. Note that the new capture will be triggered *immediately*, there is no need to use the capture button. Once you are done, use the *finish* button.

CHAPTER 5

Graphical Interface

5.1 Installation

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

Then, just re-run the *configure* step and add *gui* to your list of plugins.

5.2 Startup and Configuration

You can launch the GUI with the following command:

```
$ spread gui
```

5.3 Interface

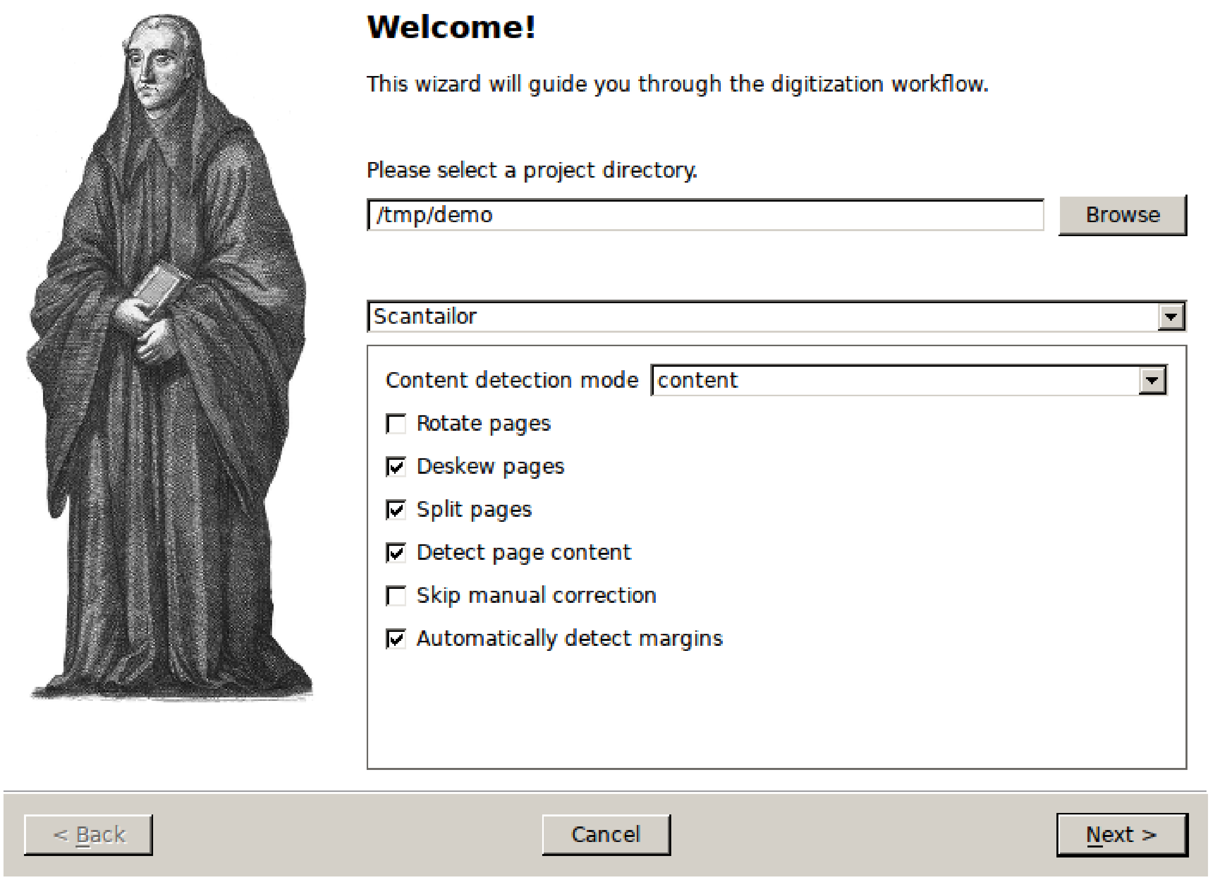
On the *first screen*, you can adjust various settings for your scan. You have to specify a project directory before you can continue. The rest of the settings depends on which plugins you have enabled. Select the plugin to configure from the dropdown menu and make your adjustments.

*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.



Welcome!

This wizard will guide you through the digitization workflow.

Please select a project directory.

/tmp/demo

Scantailor

Content detection mode: content

Rotate pages [ ]

Deskew pages [✓]

Split pages [✓]

Detect page content [✓]

Skip manual correction [ ]

Automatically detect margins [✓]

Fig. 5.1: Initial setup page

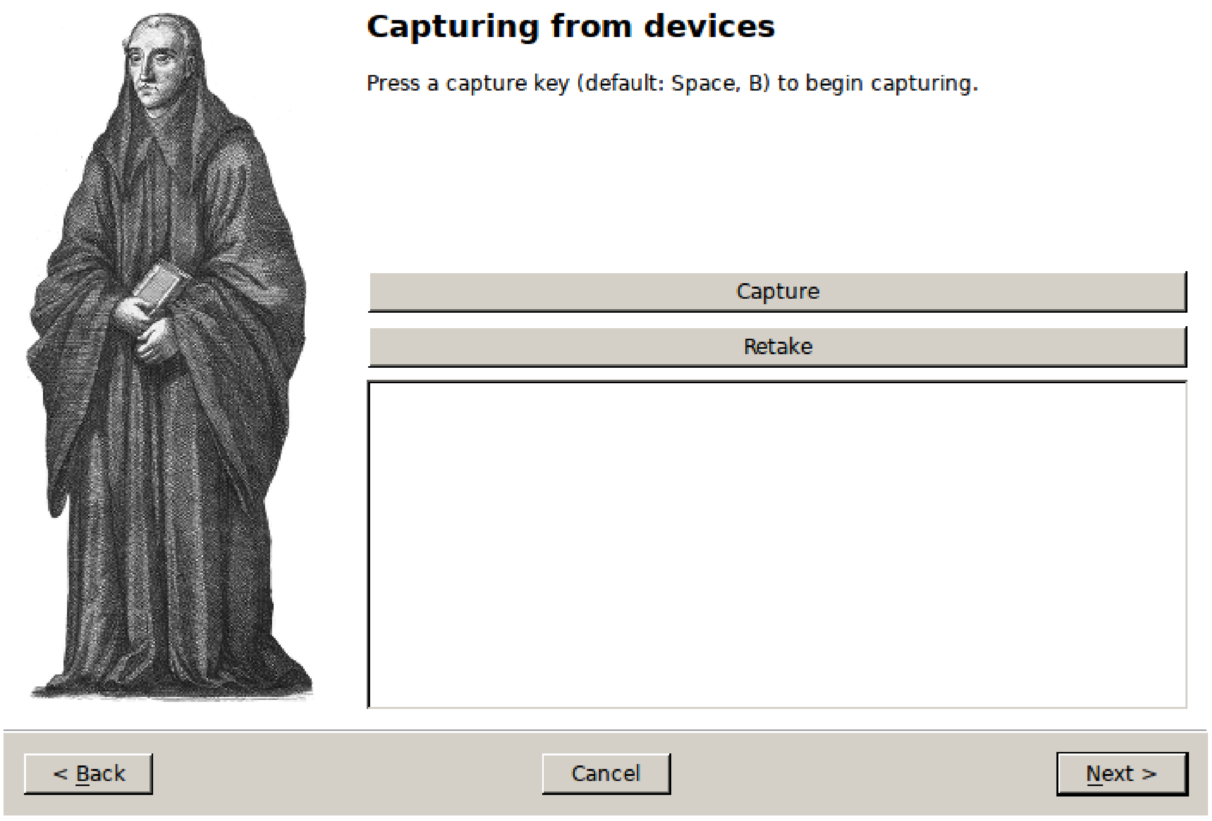


Fig. 5.2: Capture page

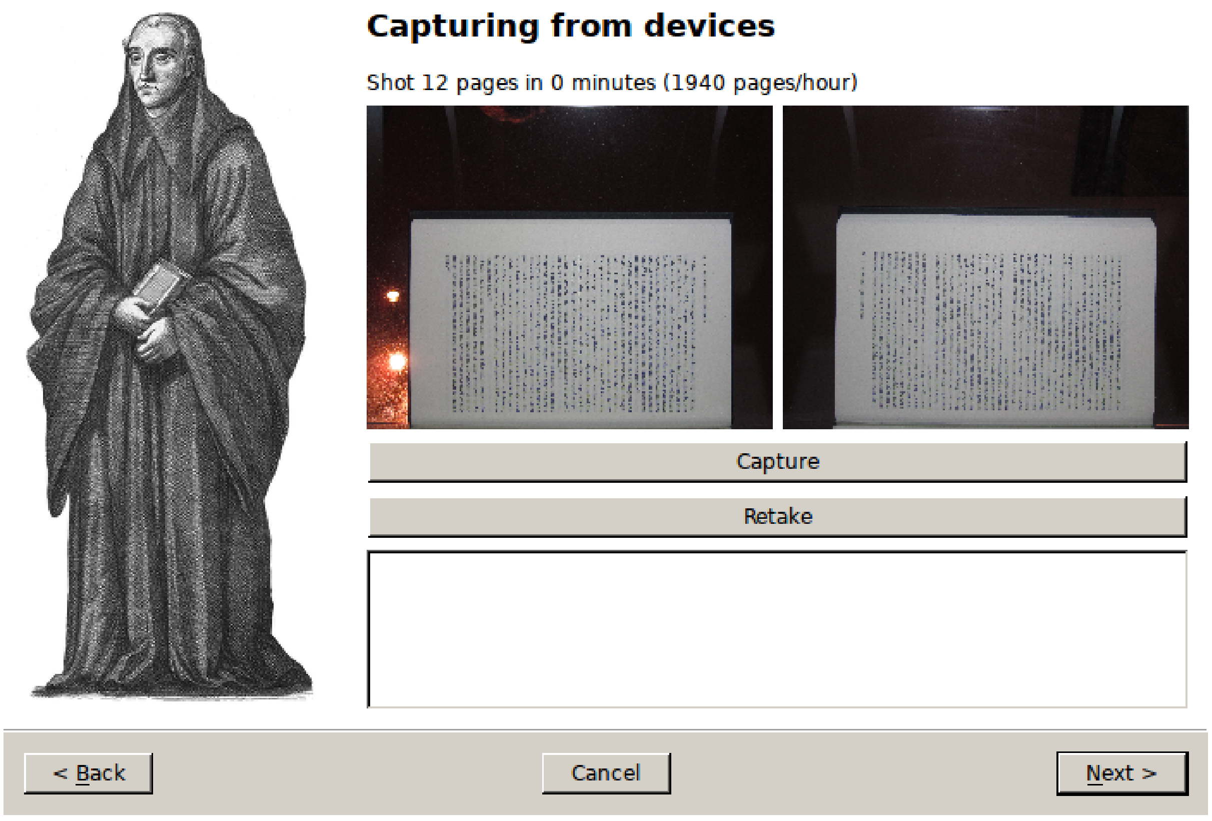
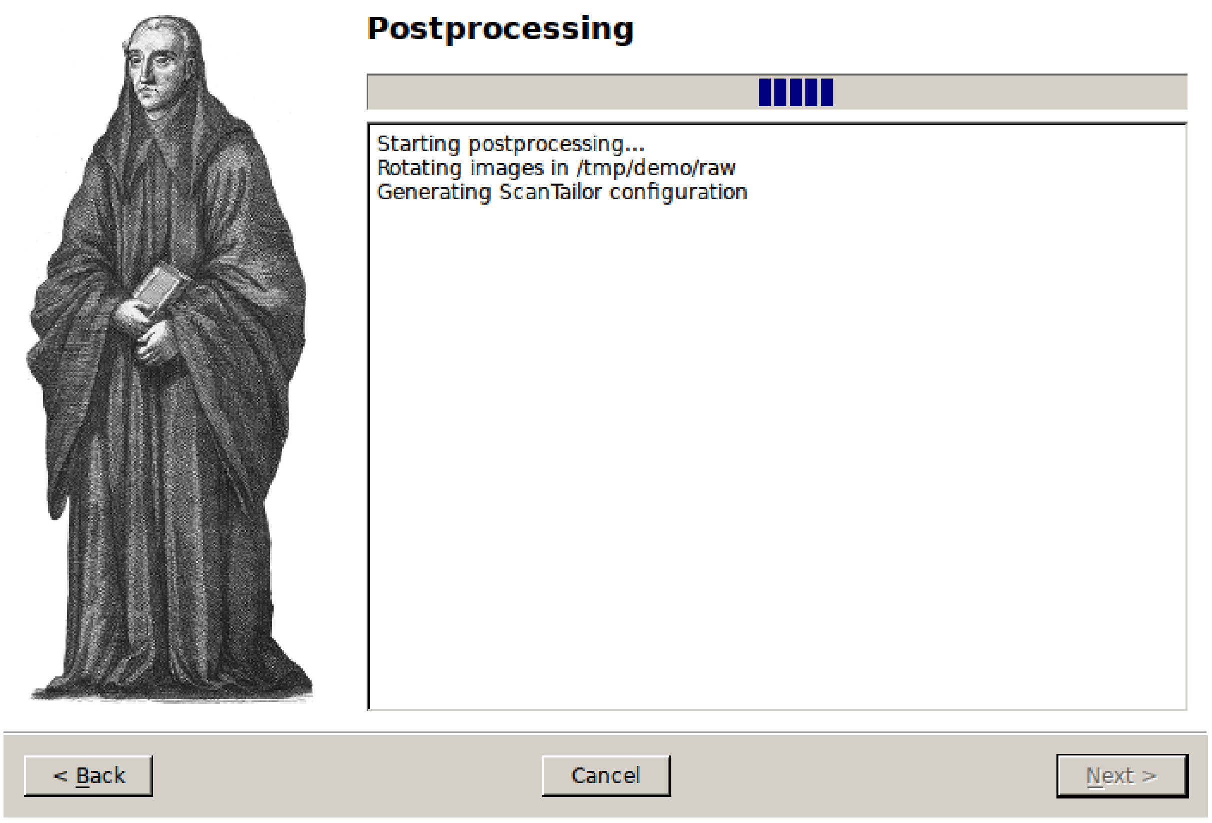
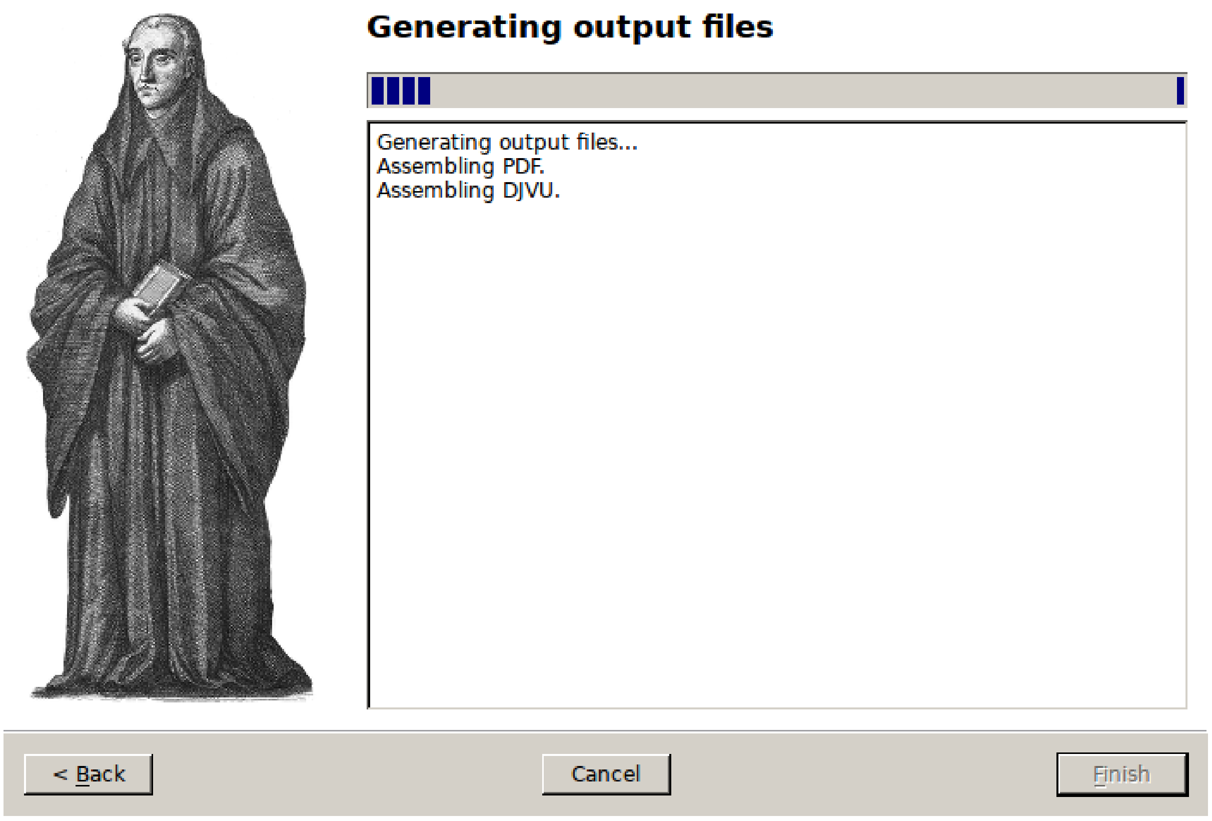


Fig. 5.3: Capture page with control images



```
Starting postprocessing...
Rotating images in /tmp/demo/raw
Generating ScanTailor configuration
```

Fig. 5.4: Postprocessing page



```
Generating output files...
Assembling PDF.
Assembling DJVU.
```

Fig. 5.5: Output page

CHAPTER 6

Command-Line Interface

6.1 Startup and Configuration

```
$ 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*. The command-line flags are the same as for the *capture*, *process* and *output* commands.

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

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.

```
$ spread postprocess <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*.

```
$ 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*.

Frequently Asked Questions

7.1 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.

... `[Error: :80: attempt to call global 'get_gui_screen_width' (a nil value)]`

spreads requires CHDK version 1.3.0 or later; you probably have the stable branch v1.2.0 installed on your camera.

CHAPTER 8

Device Drivers

In order for your capture device to work with spreads, you need to tell the application which driver it is supposed to use. This can be either done by running the *configure* subcommand and selecting one from the provided list or by manually editing the configuration file in *.config/spreads/config.yaml* in your home directory.

Currently, the following drivers are available:

8.1 chdkcamera

This driver should work with any Canon camera that runs the custom CHDK[1] firmware in version 1.3 or higher.

For it to work, the chdkptp[2] application must be installed in */usr/local/lib/chdkptp* (though that path can be configured, see below). You also need to install the *pyusb* package, with either of the following two commands:

```
$ pip install spreads[chdkcamera]
$ pip install pyusb
```

The following cameras have been tested and confirmed to work:

- A2200
- A810
- A410

If you own another CHDK-supported camera and have problems getting it to run with this driver, please open an issue on GitHub[3], we would love to make it work.

The following configuration keys/command-line flags are available:

--sensitivity `<int>`
: The ISO sensitivity value as a whole number. Default is 80.

--shutter-speed `<fraction>`
: The desired shutter speed as a fractional value. Default is 1/25. The equivalent key in the configuration file is *shutter_speed*.

[1]http://chdk.wikia.com
[2]http://www.assembla.com/spaces/chdkptp
[3]http://github.com/DIYBookScanner/spreads/issues

--zoom-level `<int>`
The desired zoom-level as a whole number. Default is 3. Make sure that this value is supported by your camera, or else you will get an error. The equivalent key in the configuration file is *zoom_level*.

--dpi `<int>`
The resolution in dots per inch that the camera captures at the given zoom level. Default is 300. You can determine this value yourself by taking a picture of an object with known dimensions, measuring its size in pixels and calculate the dots per inch from that.

--shoot-raw
Shoot RAW images instead of JPEG. Please note that this setting is **highly experimental** at the moment and RAW files are not supported by the postprocessing and output plugins as of now. The equivalent key in the configuration file is *shoot_raw*.

--focus-distance `<int/auto>`
This option allows the user to set a fixed focus distance for the cameras by specifying a whole number. This value can be obtained and automatically set in the configuration fileby running the *configure* command and following the instructions. By default, this value is set to *auto*, which means that the camera will automatically re-focus for each capture, which might give problems when there is no text or images in the center of the image. The equivalent key in the configuration file is *focus_distance*

--chdkptp-path `<path>`
Specify where the application can locate the *chdkptp* files. By default this is */usr/local/lib/chdkptp*.

8.2 gphoto2camera

This driver works with many PTP compatible camera. The full list of compatible cameras can be found here: http://www.gphoto.org/doc/remote/

For it to work, the following must be installed:

- libgphoto2: http://www.gphoto.org/ This provides the low-level PTP interface.

  You can either build from source (http://sourceforge.net/projects/gphoto/files/) or install via your local package manager (apt, brew, etc).

  For example, on Mac OS X with brew installed: $ brew install gphoto2 libgphoto2

- piggyphoto: https://github.com/YesVideo/piggyphoto This is the python interface to libgphoto2. The original source is https://github.com/alexdu/piggyphoto (our pull request to merge is pending).

  The easiest way to install is: $ pip install -e git://github.com/YesVideo/piggyphoto#egg=piggyphoto

The following cameras have been tested and confirmed to work:

- Canon T2i
- Canon 5D mk2

If you own another libgphoto2-supported camera and have problems getting it to run with this driver, please open an issue on GitHub[4], we would love to make it work.

[4]http://github.com/DIYBookScanner/spreads/issues

The following configuration keys/command-line flags are available:

`--iso` `<string>`
: The ISO value. Default is 'Auto'.

`--shutter-speed` `<fraction>`
: The desired shutter speed as a fractional value. Default is 1/25. The equivalent key in the configuration file is *shutter_speed*.

`--aperture` `<float>`
: The desired aperture expressed as an f-stop (without the 'f/' prefix). Default is 5.6. The equivalent key in the configuration file is *aperture*.

`--shoot-raw`
: Shoot RAW images instead of JPEG. Please note that this setting is **highly experimental** at the moment and RAW files are not supported by the postprocessing and output plugins as of now. The equivalent key in the configuration file is *shoot_raw*.

CHAPTER 9

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.

9.1 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.

9.1.1 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.

9.1.2 web

Launches the spread web interface that offers a REST-ish API with which you can control the application from any HTTP client. It also includes a client-side JavaScript application that can be used from any recent browser (Firefox or Chrome recommended). Fore more details, consult the *Web interface documentation <web_doc>* and the *REST API documentation <rest_api>*

`--standalone-device`
: Enable standalone mode. This option can be used for devices that are dedicated to scanning (e.g. a RaspberryPi that runs spreads and nothing else). At the moment the only additional feature it enables is the ability to shutdown the device from the web interface and REST API.

`--debug`
: Run the application debugging mode.

`--project-dir` `<path>`
: Location where workflow files are stored. By default this is *~/scans*.

`--mode` `[scanner,` **`processor,`** **`full`** `(default)]`
: Select the mode the web plugin is supposed to run in. scanner: Only offer components neccessary for capture and download/submission to a postprocessing server processor:

Start as a postprocessing server that can receive workflows over the network from other 'scanner' instances full: Combines the above two modes, allows for capture and postprocessing/output generation on the same machine

`--port` `<port> (default: 5000)`
: Select port on which the web plugin is supposed to listen on

9.2 *postprocess* plugins

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

9.2.1 autorotate

Automatically rotates the images according to their device of origin.

9.2.2 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.

9.2.3 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*.

9.3 *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*.

9.3.1 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.

9.3.2 djvubind

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

See also:

Extending spreads functionality

CHAPTER 10

Contributing

CHAPTER 11

Extending *spreads*

11.1 Setting up a development environment

The easiest way to work on spreads is to install it to an editable virtual Python environment using the `virtualenv` tool and installing spreads into it using `pip` with the `-e` option. This option allows the virtual environment to treat a spreads repository checked out from git as a live installation.

For example, on a Debian-based system, assuming the git repository for spreads is checked out to `./spreads`:

```
virtualenv spreadsenv
cd spreadsenv
source ./bin/activate
# The following dependencies are not pulled in automatically by
# setuptools
pip install cffi
pip install jpegtran-cffi
pip install -e ../spreads
```

Other prerequisite packages you may require include:

> libffi-dev libjpeg8-dev libturbojpeg

11.2 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[1] and the relevant part of spreads' setup.py[2] for a reference implementation.

Devices have to implement a *yield_devices<spreads.plugin.DevicePlugin.yield_devices>* method that scans the system for supported devices and returns fully instantiated device objects for those.

[1]https://github.com/DIYBookScanner/spreads/blob/master/spreadsplug/dev/chdkcamera.py
[2]https://github.com/DIYBookScanner/spreads/blob/master/setup.py

11.3 Declaring available configuration options for plugins

Device drivers (as well as all plugins) can implement the *configuration_templates<spreads.plugin.SpreadsPlugin.configuration_template>* method that returns a dictionary of setting keys and *PluginOption<spreads.plugin.PluginOption>* objects. These options will be visible across all supported interfaces and also be read from the configuration file and command-line arguments.

11.4 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 one of the available mixin classes (at the moment these are *CaptureHooksMixin<spreads.plugin.CaptureHooksMixin>*, *TriggerHooksMixin<spreads.plugin.TriggerHooksMixin>*, *ProcessHookMixin<spreads.plugin.ProcessHookMixin>*, *OutputHookMixin<spreads.plugin.OutputHookMixin>*). You then have to implement each of the required methods for the mixins of your choice.

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[3]

See also:

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

11.5 Adding new commands

You can also add entirely new commands to the application. Simply subclass `HookPlugin` and *SubcommandHookMixin<spreads.plugin.SubcommandHookMixin>*, implement the `add_command_parser` classmethod and add your new class as an entry point to the `spreadsplug.hooks` namespace. See the web[4] and gui[5] plugins for examples of plugins that add custom subcommands.

[3]https://github.com/DIYBookScanner/spreads/blob/master/spreadsplug
[4]https://github.com/DIYBookScanner/spreads/blob/master/spreadsplug/web/__init__.py
[5]https://github.com/DIYBookScanner/spreds/blob/master/spreadsplug/gui/__init__.py

CHAPTER 12

API Reference

12.1 spreads API Reference

spreads package

This is the core package for spreads. Except for the `spreads.cli` and `spreads.main` modules (which contain the logic for the *spread* command-line application) everything in this package is UI-agnostic and designed to be used from plugins in the *spreadsplug* namespace.

It includes the following modules (in no particular order):

`spreads.main` Core logic for application startup and parsing of command-line arguments

`spreads.cli` Implementation of the command-line interface, i.e. the *configure*, *capture*, *postprocess*, *output* and *wizard* subcommands.

`spreads.config` Classes for working with configuration, both per-workflow and application-wide. Most important for plugin developers is the `spreads.config.OptionTemplate` class, which allows for the UI-agnostic declaration of configuration options.

`spreads.workflow` This is by far the largest module in the core and contains the `spreads.workflow.Workflow` class that is the central entity in the application. Also included are classes for representing single page entities and TOC-entries, as well as various signals that can be emitted by a workflow entity.

`spreads.metadata` Contains the `spreads.metadata.Metadata` entity class that manages the reading and writing of metadata values.

`spreads.plugin` The most important module for plugin authors. It contains the various interfaces (all inheriting from `spreads.plugin.SpreadsPlugin`) that plugins and device drivers can implement, as well as functions (intended for use by the core) to enumerate and initialize plugins and device drivers.

`spreads.util` Various helper functions that can be useful for both plugin authors and the core. Also contains the various `Exception` subclasses used throughout the core and the plugin interface.

`spreads.tkconfigure` Implementation of the graphical configuration dialog (accessible via the *guiconfigure* subcommand), using the Tkinter bindings from Python's standard library.

Public plugin API (realized through a range of abstract classes) and utility functions for enumerating and loading plugins.

exception `spreads.plugin.`**`ExtensionException`**(*message=None, extension=None*)
" Raised when something went wrong during plugin enumeration/ or instantiation.

`__init__`(*message=None, extension=None*)

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

`on_progressed` **= <blinker.base.NamedSignal object at 0x7fa6ecc488d0; u'plugin:progressed'>**

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 Option-Template object for each key.

Example:

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

Returns dict with *unicode* -> *`spreads.config.OptionTemplate`*

`__init__`(*config*)
Initialize the plugin.

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

class `spreads.plugin.`**`DeviceFeatures`**
Enum that provides various constants that *`DeviceDriver`* implementations can expose in their *`DeviceDriver.features`* tuple to declare support for one or more given features.

`PREVIEW` **= <DeviceFeatures.PREVIEW: 1>**
Device can grab a preview picture

`IS_CAMERA` **= <DeviceFeatures.IS_CAMERA: 2>**
Device class allows the operation of two devices simultaneously (mainly to be used by cameras, where each device is responsible for capturing a single page.

`CAN_DISPLAY_TEXT` **= <DeviceFeatures.CAN_DISPLAY_TEXT: 3>**
Device can display arbitrary messages on its screen

`CAN_ADJUST_FOCUS` **= <DeviceFeatures.CAN_ADJUST_FOCUS: 4>**
Device can read set its own focus distance and read out its autofocus

class `spreads.plugin.`**`DeviceDriver`**(*config, device*)
Base class for device drivers.

Subclass to implement support for different devices.

features = ()
: Tuple of `DeviceFeatures` constants that designate the features the device offers.

classmethod configuration_template()
: Returns some pre-defined options when the implementing devices has the `DeviceFeatures.IS_CAMERA` feature.

__init__(*config*, *device*)
: Set connection information and other properties.

 Parameters

 - **config** (`spreads.confit.ConfigView`) – spreads configuration
 - **device** (py:class:*usb.core.Device*) – USB device to use for the object

connected()
: Check if the device is still connected.

 Return type bool[1]

set_target_page(*target_page*)
: Set the device target page, if applicable.

 Parameters **target_page** (unicode, one of *odd* or *even*) – The target page

prepare_capture()
: 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 and applying all relevant settings.

capture(*path*)
: Capture a single image with the device.

 Parameters **path** (`pathlib.Path`[2]) – Path for the image

finish_capture()
: Tell device to finish capturing.

 What this means exactly is up to the implementation and the type of device, with a camera it could e.g. involve retracting the lense.

update_configuration(*updated*)
: Update the device configuration.

 The implementing device driver should propagate these updates to the hardware and make sure everything is applied correctly.

 Parameters **updated** (*dict*[3]) – Updated configuration values

on_progressed = <blinker.base.NamedSignal object at 0x7fa6ecc488d0; u'plugin:progressed'>

class `spreads.plugin.`**HookPlugin**(*config*)
: Base class for HookPlugins.

 Implement one of the available mixin classes (`SubcommandHooksMixin`, `CaptureHooksMixin`, py:class:*TriggerHooksMixin*, `ProcessHooksMixin`, `OutputHooksMixin`) to register for the appropriate hooks.

[1]http://docs.python.org/2.7/library/functions.html#bool
[2]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[3]http://docs.python.org/2.7/library/stdtypes.html#dict

`__init__`(*config*)
Initialize the plugin.

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

`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 Option-Template object for each key.

Example:

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

Returns dict with *unicode* -> *`spreads.config.OptionTemplate`*

`on_progressed` = **<blinker.base.NamedSignal object at 0x7fa6ecc488d0; u'plugin:progressed'>**

class `spreads.plugin.`**`SubcommandHooksMixin`**
Mixin for plugins that want to provide custom subcommands.

`__init__`
x.__init__(...) initializes x; see help(type(x)) for signature

class `spreads.plugin.`**`CaptureHooksMixin`**
Mixin for plugins that want to hook into the capture process.

`prepare_capture`(*devices*)
Perform some action before capturing begins.

Parameters **`devices`** (list of *`DeviceDriver`*) – The devices used for capturing

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

Parameters

- **`devices`** (list of *`DeviceDriver`*) – The devices used for capturing
- **`path`** (`pathlib.Path`[4]) – Workflow path

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

Parameters

- **`devices`** (list of *`DeviceDriver`*) – The devices used for capturing

[4]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path

- **path** (pathlib.Path[5]) – Workflow path

__init__
x.__init__(...) initializes x; see help(type(x)) for signature

class spreads.plugin.**TriggerHooksMixin**
Mixin for plugins that want to provice customized ways of triggering a capture.

start_trigger_loop(*capture_callback*)

Start a thread that runs an event loop and periodically triggers a capture by calling the *capture_callback*.

Parameters capture_callback (*function*) – The function that triggers a capture

stop_trigger_loop()
Stop the thread started by *start_trigger_loop()*.

__init__
x.__init__(...) initializes x; see help(type(x)) for signature

class spreads.plugin.**ProcessHooksMixin**
Mixin for plugins that want to provide postprocessing functionality.

process(*pages, target_path*)

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

Parameters

- **pages** (list of *spreads.workflow.Page*) – Pages to be processed
- **target_path** (pathlib.Path[6]) – Target directory for processed files

__init__
x.__init__(...) initializes x; see help(type(x)) for signature

class spreads.plugin.**OutputHooksMixin**
Mixin for plugins that want to create output files.

output(*pages, target_path, metadata, table_of_contents*)
Assemble an output file from the pages.

Parameters

- **pages** (list of *spreads.workflow.Page*) – Project path
- **target_path** (pathlib.Path[7]) – Target directory for processed files
- **metadata** (*spreads.metadata.Metadata*) – Metadata for workflow

[5] http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[6] http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[7] http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path

- **table_of_contents** (list of *spreads.workflow.TocEntry*) – Table of Contents for workflow

__init__
x.__init__(...) initializes x; see help(type(x)) for signature

`spreads.plugin.`**`available_plugins`**`()`
Get the names of all installed plugins.

Returns List of plugin names

`spreads.plugin.`**`get_plugins`**`(`**names*`)`
Get instantiated and configured plugin instances.

Parameters **names** (*unicode*[8]) – One or more plugin names

Returns Mapping of plugin name to plugin instance

Return type dict of unicode -> *SpreadsPlugin*

`spreads.plugin.`**`available_drivers`**`()`
Get the names of all installed device drivers.

Returns List of driver names

`spreads.plugin.`**`get_driver`**`(`*driver_name*`)`
Get a device driver.

Parameters **driver_name** (*unicode*[9]) – Name of driver to instantiate

Returns The driver class

Return type *DeviceDriver* class

`spreads.plugin.`**`get_devices`**`(`*config, force_reload=False*`)`
Get initialized and configured device instances.

Parameters

- **config** (*spreads.config.Configuration*) – Global configuration
- **force_reload** (*bool*[10]) – Don't load devices from cache

Returns Device instances

Return type list of *DeviceDriver* objects

Central *Workflow* entity (and its signals) and various associated entities.

exception `spreads.workflow.`**`ValidationError`**`(`*message=None, **kwargs*`)`
Raised when some kind of validation error occured.

Attr message General error message

Attr errors Mapping from field name to validation error message

__init__(*message=None, **kwargs*)
Create new instance.

`**kwargs` should be a mapping from a field name to an error message.

[8]http://docs.python.org/2.7/library/functions.html#unicode
[9]http://docs.python.org/2.7/library/functions.html#unicode
[10]http://docs.python.org/2.7/library/functions.html#bool

class `spreads.workflow.`**`Page`**(*raw_image*, *sequence_num=None*, *capture_num=None*, *page_label=None*, *processed_images=None*)
Entity that holds information about a single page.

Attr raw_image The path to the raw image.

Attr processed_images A dictionary of plugin names mapped to the path of a processed file.

Attr capture_num The capture number of the page, i.e. at what position in the workflow it was recorded, including aborted and retaken shots.

Attr sequence_num The sequence number of the page, i.e. at what position in the list of 'good' captures it is. Usually identical with the position in the containing *pages* list. Defaults to the capture number.

Attr page_label A label for the page. Must be an integer, a string of digits or a roman numeral (e.g. 12, '12', 'XII'). Defaults to the sequence number.

`__init__`(*raw_image*, *sequence_num=None*, *capture_num=None*, *page_label=None*, *processed_images=None*)

`get_latest_processed`(*image_only=True*)
Get the least recent postprocessed file

Parameters **`image_only`** (*bool*[11]) – Only return image files (e.g. no OCR files)

Returns Path to least recent postprocessed file

Return type `pathlib.Path`[12]

`to_dict`()
Serialize entity to a dict.

Used by *`spreads.util.CustomJSONEncoder`*.

class `spreads.workflow.`**`TocEntry`**(*title*, *start_page*, *end_page*, *children=None*)
Represent a 'table of contents' entry.

Attr title Label/title of the entry

Attr start_page First page of the entry

Attr end_page First page no longer part of the entry

:attr children; Other *`TocEntry`* objects that designate a sub-range of this entry

`__init__`(*title*, *start_page*, *end_page*, *children=None*)

`to_dict`()
Serialize entity to a dict.

Used by *`spreads.util.CustomJSONEncoder`*.

class `spreads.workflow.`**`Workflow`**(*path*, *config=None*, *metadata=None*)
Core entity for managing scanning workflows.

Attr id UUID for the workflow

[11] http://docs.python.org/2.7/library/functions.html#bool
[12] http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path

Attr status Current status. Keys are `step` ('capture', 'process' or 'output'), `step_progress` (Progress as a value between 0 and 1) and `prepared` (whether capture is already prepared).

Attr path Path to directory containing the workflow's data.

:type path; `pathlib.Path`[13] :attr bag: Underlying BagIt data structure :type bag: py:class:*spreads.vendor.bagit.Bag* :attr slug: ASCIIfied version of workflow title without spaces. :attr config: Configuration for the worklfow, takes precedence

over the global configuration).

Attr metadata Metadata, contains at least a `title` field.

Attr pages Pages available in the workflow

Attr table_of_contents Table of contents entries in the workflow

Attr last_modified Time of last modification

Attr devices Active devices

Attr out_files Generated output files

classmethod `create`(*location*, *metadata=None*, *config=None*)
Create a new Workflow.

Parameters

- **`location`** (unicode or `pathlib.Path`[14]) – Base directory that the workflow should be created in
- **`metadata`** (*dict*[15]) – Initial metadata for workflow. Must at least contain a *title* item.
- **`config`** (dict or *`spreads.config.Configuration`*) – Initial configuration for workflow

Returns The new instance

Return type *`Workflow`*

classmethod `find_all`(*location*, *key=u'slug'*, *reload=False*)
List all workflows in the given location.

Parameters

- **`location`** (unicode or `pathlib.Path`[16]) – Location where the workflows are located
- **`key`** (*str/unicode*) – Attribute to use as key for returned dict
- **`reload`** (*bool*[17]) – Do not load workflows from cache

Returns All found workflows

[13]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[14]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[15]http://docs.python.org/2.7/library/stdtypes.html#dict
[16]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[17]http://docs.python.org/2.7/library/functions.html#bool

Return type dict[18]

classmethod `find_by_id`(*location*, *id*)
Try to locate a workflow with the given id in a directory.

Parameters

- **`location`** (unicode or `pathlib.Path`[19]) – Base directory that contains workflows to be searched among
- **`id`** – ID of workflow to be searched for

Return type *`Workflow`* or None

classmethod `find_by_slug`(*location*, *slug*)
Try to locate a workflow that matches a given slug in a directory.

Parameters

- **`location`** (unicode or `pathlib.Path`[20]) – Base directory that contains workflows to be searched among
- **`slug`** (*unicode*[21]) – Slug of workflow to be searched for

Return type *`Workflow`* or None

classmethod `remove`(*workflow*)
Delete a workflow from the disk and cache.

Parameters **`workflow`** (*`Workflow`*) – Workflow to be deleted

`__init__`(*path*, *config=None*, *metadata=None*)

`remove_pages`(**pages*)
Remove one or more pages from the workflow.

This will irrevocably remove the page metadata as well as all of its associated files, so use responsibly!

Parameters **`pages`** (*`Page`*) – One or more pages to remove

`crop_page`(*page*, *left*, *top*, *width=None*, *height=None*, *async=False*)
Crop a page's raw image.

Parameters

- **`page`** – Page the raw image of which should be cropped
- **`left`** – X coordinate of crop boundary
- **`top`** – Y coordinate of crop boundary
- **`width`** – Width of crop box
- **`height`** – Height of crop box
- **`async`** – Perform the cropping in a background thread

Returns The Future object when `async` was `True`

[18]http://docs.python.org/2.7/library/stdtypes.html#dict
[19]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[20]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[21]http://docs.python.org/2.7/library/functions.html#unicode

Return type `concurrent.futures.Future`

`save`()
Persist all changes to the corresponding files on disk.

`prepare_capture`()
Prepare capture on devices and initialize trigger plugins.

`finish_capture`()
Wrap up capture process.

`process`()
Run all captured pages through post-processing.

`output`()
Assemble pages into output files.

`update_configuration`(*values*)
Update the workflow's configuration.

Metadata class and utility functions.

`get_isbn_suggestions()` and `get_isbn_metadata()` return a dictionary with the following keys (which corresponds to the Dublin Core field of the same name): *creator*, *identifier*, *date*, *language*.

`spreads.metadata.`**`get_isbn_suggestions`**(*query*)
For a given *query*, return a list of metadata suggestions.

Parameters **`query`** (*unicode*[22]) – Search query

Returns List of suggestions

Return type list of dict

`spreads.metadata.`**`get_isbn_metadata`**(*isbn*)

For a given valid ISBN number (-10 or -13) return the corresponding metadata.

Parameters **`isbn`** (*unicode*[23]) – A valid ISBN-10 or ISBN-13

Returns Metadata for ISBN

Return type dict or *None* if ISBN is not valid or does not exist

class `spreads.metadata.`**`SchemaField`**(*key*, *description=None*, *multivalued=False*)
Definition of a field in a metadata schema.

Attr key Key/field name

Attr description Description of the field

Attr multivalued Whether the field can hold multiple values

`__init__`(*key*, *description=None*, *multivalued=False*)

class `spreads.metadata.`**`Metadata`**(*base_path*)
dict-like object that has a schema of metadata fields (currently hard-wired to Dublin Core) and persists all operations to a *dcmeta.txt* text file on the disk.

[22] http://docs.python.org/2.7/library/functions.html#unicode
[23] http://docs.python.org/2.7/library/functions.html#unicode

`__init__`(*base_path*)

Create a new instance and try to load current values from an existing file.

Parameters **`base_path`** – Directory where *dcmeta.txt* should be stored

Configuration entities.

class `spreads.config.`**`OptionTemplate`**(*value*, *docstring=None*, *selectable=False*, *advanced=False*, *depends=None*)

Definition of a configuration option.

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

Attr docstring A string explaining the configuration option

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

Attr advanced Whether the option is an advanced option

Attr depends Make option dependant of some other setting (if passed a dict) or another plugin (if passed a string)

`__init__`(*value*, *docstring=None*, *selectable=False*, *advanced=False*, *depends=None*)

class `spreads.config.`**`Configuration`**(*appname=u'spreads'*)

Entity managing configuration state.

Uses `confit.Configuration` underneath the hood and follows its 'overlay'-principle. Proxies `__getitem__()` and `__setitem__()` from it, so it can be used as a dict-like type.

`__init__`(*appname=u'spreads'*)

Create new instance and load default and current configuration.

Parameters **`appname`** – Application name, configuration will be loaded from this name's default configuration directory

`keys`()

See `confit.ConfigView.keys()`

`dump`(*filename=None*, *full=True*, *sections=None*)

See `confit.Configuration.dump()`

`flatten`()

See `confit.Configuration.flatten()`

`load_templates`()

Get all available configuration templates from the activated plugins.

Returns Mapping from plugin name to template mappings.

Return type dict unicode -> (dict unicode -> `OptionTemplate`)

`cfg_path`

Path to YAML file of the user-specific configuration.

Returns Path

Return type `pathlib.Path`[24]

with_overlay(*overlay*)
Get a new configuration that overlays the provided configuration over the present configuration.

Parameters **overlay** (`confit.ConfigSource` or dict) – The configuration to be overlaid

Returns A new, merged configuration

Return type `confit.Configuration`

as_view()
Return the *Configuration* as a `confit.ConfigView` instance.

load_defaults(*overwrite=True*)
Load default settings from option templates.

Parameters **overwrite** – Whether to overwrite already existing values

set_from_template(*section, template, overwrite=True*)
Set default options from templates.

Parameters

- **section** (*unicode*[25]) – Target section for settings
- **overwrite** – Whether to overwrite already existing values

set_from_args(*args*)
Apply settings from parsed command-line arguments.

Parameters **args** (`argparse.Namespace`[26]) – Parsed command-line arguments

Various utility functions and classes.

exception spreads.util.**SpreadsException**
General exception

exception spreads.util.**DeviceException**
Raised when a device-related error occured.

exception spreads.util.**MissingDependencyException**
Raised when a dependency for a plugin is missing.

spreads.util.**get_version**()
Get installed version via pkg_resources.

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

Parameters **name** (*unicode*[27]) – name of the executable

Returns Path to executable or None if not found

Return type unicode or None

[24]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path
[25]http://docs.python.org/2.7/library/functions.html#unicode
[26]http://docs.python.org/2.7/library/argparse.html#argparse.Namespace
[27]http://docs.python.org/2.7/library/functions.html#unicode

`spreads.util.`**`is_os`**(*osname*)

Check if the current operating system matches the expected.

Parameters **`osname`** – Operating system name as returned by `platform.system()`[28]

Returns Whether the OS matches or not

Return type bool[29]

`spreads.util.`**`check_futures_exceptions`**(*futures*)

" Go through passed `concurrent.futures._base.Future` objects and re-raise the first Exception raised by any one of them.

Parameters **`futures`** (iterable with `concurrent.futures._base.Future` instances) – Iterable that contains the futures to be checked

`spreads.util.`**`get_free_space`**(*path*)

Return free space on file-system underlying the passed path.

Parameters **`path`** – Path on file-system the free space of which is desired.

:type path; unicode :return: Free space in bytes. :rtype: int

`spreads.util.`**`get_subprocess`**(*cmdline*, ***kwargs*)

Get a `subprocess.Popen`[30] instance.

On Windows systems, the process will be ran in the background and won't open a cmd-window or appear in the taskbar. The function signature matches that of the `subprocess.Popen`[31] initialization method.

`spreads.util.`**`wildcardify`**(*pathnames*)

Try to generate a single path with wildcards that matches all *pathnames*.

Parameters **`pathnames`** – List of pathnames to find a wildcard string for

Returns The wildcard string or None if none was found

Return type unicode or None

`spreads.util.`**`diff_dicts`**(*old*, *new*)

Get the difference between two dictionaries.

Parameters

- **`old`** (*dict*[32]) – Dictionary to base comparison on
- **`new`** (*dict*[33]) – Dictionary to compare with

Returns A (possibly nested) dictionary containing all items from *new* that differ from the ones in *old*

[28]http://docs.python.org/2.7/library/platform.html#platform.system
[29]http://docs.python.org/2.7/library/functions.html#bool
[30]http://docs.python.org/2.7/library/subprocess.html#subprocess.Popen
[31]http://docs.python.org/2.7/library/subprocess.html#subprocess.Popen
[32]http://docs.python.org/2.7/library/stdtypes.html#dict
[33]http://docs.python.org/2.7/library/stdtypes.html#dict

Return type dict[34]

`spreads.util.`**`slugify`**(*text*, *delimiter=u'-'*)

Generates an ASCII-only slug.

Code adapted from Flask snipped by Armin Ronacher: http://flask.pocoo.org/snippets/5/

Parameters

- **`text`** (*unicode*[35]) – Text to create slug for
- **`delimiter`** (*unicode*[36]) – Delimiter to use in slug

Returns The generated slug

Return type unicode[37]

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 StreamHandler

Kudos to Leigh MacDonald: http://goo.gl/Lpr6C5

`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[38]

class `spreads.util.`**`EventHandler`**(*level=0*)

Subclass of `logging.Handler` that emits a `blinker.base.Signal` whenever a new record is emitted.

`on_log_emit` **= <blinker.base.NamedSignal object at 0x7fa6ee644210; u'logrecord'>**

`spreads.util.`**`get_data_dir`**(*create=False*)

Return (and optionally create) the user's default data directory.

Parameters **`create`** (*bool*[39]) – Create the data directory if it doesn't exist

Returns Path to the default data directory

Return type unicode[40]

[34]http://docs.python.org/2.7/library/stdtypes.html#dict
[35]http://docs.python.org/2.7/library/functions.html#unicode
[36]http://docs.python.org/2.7/library/functions.html#unicode
[37]http://docs.python.org/2.7/library/functions.html#unicode
[38]http://docs.python.org/2.7/library/functions.html#bool
[39]http://docs.python.org/2.7/library/functions.html#bool
[40]http://docs.python.org/2.7/library/functions.html#unicode

spreads.util.**colorize**(*text*, *color*)

Return text with a new ANSI foreground color.

Parameters

- **text** – Text to be wrapped
- **color** (str (from *colorama.ansi <http://git.io/9qnt0Q>*)) – ANSI color to wrap text in

Returns Colorized text

class spreads.util.**RomanNumeral**(*value*, *case=u'upper'*)

Number type that represents integers as Roman numerals and that can be used in all arithmetic operations applicable to integers.

static **is_roman**(*value*)

Check if *value* is a valid Roman numeral.

Parameters **value** (*unicode*[41]) – Value to be checked

Returns Whether the value is valid or not

Return type bool[42]

__init__(*value*, *case=u'upper'*)

Create a new instance.

Parameters **value** (int, unicode containing valid Roman numeral or RomanNumeral) – Value of the instance

class spreads.util.**CustomJSONEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Custom json.JSONEncoder[43].

Uses an object's *to_dict* method if present for serialization.

Serializes pathlib.Path[44] instances to the string representation of their relative path to a BagIt-compliant directory or their absolute path if not applicable.

Core logic for application startup and parsing of command-line arguments

spreads.main.**add_argument_from_template**(*extname*, *key*, *template*, *parser*, *current_val*)

Add option from *template* to *parser* under the name *key*.

Templates with a boolean value type will create a *–<key>* or *–no-<key>* flag, depending on their current value.

Parameters

- **extname** – Name of the configuration section this option's result should be stored in

[41]http://docs.python.org/2.7/library/functions.html#unicode
[42]http://docs.python.org/2.7/library/functions.html#bool
[43]http://docs.python.org/2.7/library/json.html#json.JSONEncoder
[44]http://pathlib.readthedocs.org/en/pep428/index.html#pathlib.Path

- **key** – Configuration key in section, will also determine the name of the argument.
- **template** (*spreads.config.OptionTemplate*) – Template for the argument
- **parser** (argparse.ArgumentParser[45]) – Argument parser the argument should be added to
- **current_val** – Current value of the option

spreads.main.**main**()

Entry point for *spread* command-line application.

spreads.main.**run**()

Setup the application and run subcommand

spreads.main.**run_config_windows**()

Entry point to launch graphical configuration dialog on Windows.

spreads.main.**run_service_windows**()

Entry point to launch web plugin server on Windows.

spreads.main.**setup_logging**(*config*)

Conigure application-wide logger.

Parameters **config** (*spreads.config.Configuration*) – Global configuration

spreads.main.**setup_parser**(*config*)

Sets up an argparse.ArgumentParser[46] instance with all options and subcommands that are available in the core and activated plugins.

Parameters **config** (*spreads.config.Configuration*) – Current application configuration

Returns Fully initialized argument parser

Return type argparse.ArgumentParser[47]

spreads.main.**should_show_argument**(*template*, *active_plugins*)

Checks the spreads.config.OptionTemplate.depends attribute for dependencies on other plugins and validates them against the list of activated plugins.

We do not validate dependencies on other configuration settings because we don't have access to the final state of the configuration at this time, since the configuration can potentially be changed by other command-line flags.

Parameters

- **template** (*spreads.config.OptionTemplate*) – Template to check
- **active_plugins** – List of names of activated plugins

Returns Whether or not the argument should be displayed

Command-Line interface for configuration, capture, output and postprocessing.

[45]http://docs.python.org/2.7/library/argparse.html#argparse.ArgumentParser
[46]http://docs.python.org/2.7/library/argparse.html#argparse.ArgumentParser
[47]http://docs.python.org/2.7/library/argparse.html#argparse.ArgumentParser

`spreads.cli.`**`getch`**`()`

Waits for a single character to be entered on stdin and returns it.

Returns Character that was entered

Return type str[48]

`spreads.cli.`**`draw_progress`**`(`*progress*`)`

Draw a progress bar to stdout.

Parameters **`progress`** (*float*[49]) – Progress value between 0 and 1

`spreads.cli.`**`configure`**`(`*config*`)`

Configuration subcommand that runs through the various dialogs, builds a new configuration and writes it to disk.

Parameters **`config`** (*`spreads.config.Configuration`*) – Currently active global configuration

`spreads.cli.`**`capture`**`(`*config*`)`

Dialog to run through the capture process.

Parameters **`config`** (*`spreads.config.Configuration`*) – Currently active global configuration

`spreads.cli.`**`postprocess`**`(`*config*`)`

Launch postprocessing plugins and display their progress

Parameters **`config`** (*`spreads.config.Configuration`*) – Currently active global configuration

`spreads.cli.`**`output`**`(`*config*`)`

Launch output plugins and display their progress

Parameters **`config`** (*`spreads.config.Configuration`*) – Currently active global configuration

`spreads.cli.`**`wizard`**`(`*config*`)`

Launch every step in succession with the same configuration.

Parameters **`config`** (*`spreads.config.Configuration`*) – Currently active global configuration

Graphical configuration dialog.

class `spreads.tkconfigure.`**`TkConfigurationWindow`**`(`*spreads_config*, *master=None*`)`

Window that holds the dialog

`__init__``(`*spreads_config*, *master=None*`)`

Initialize Window with global configuration.

Parameters **`spreads_config`** (*`spreads.config.Configuration`*) – Global configuration

`update_plugin_config``(`*plugins*`)`

Update list of activated plugins and load its default configuration.

[48]http://docs.python.org/2.7/library/functions.html#str

[49]http://pillow.readthedocs.org/reference/ImageMath.html#float

Parameters **`plugins`** (*list of unicode*) – List of names of plugins to activate

`on_update_driver`(*event*)
Callback for when the user selects a driver.

Updates the driver in the configuration and toggles the status of widgets that depend on certain device features.

Parameters **`event`** (`Tkinter.Event`) – Event from Tkinter

`on_update_plugin_selection`(*event*)
Callback for when the user toggles a plugin.

Tries to load the newly selected plugins. If loading fails, a dialog with the cause of failure will be displayed and the plugin will be highlighted in the list and made inactive. If successful, the plugin will be added to the 'postprocessing order' widget (if it implements *`spreads.plugin.ProcessHooksMixin`*) and the configuration will be updated.

Parameters **`event`** (`Tkinter.Event`) – Event from Tkinter

`on_process_plugin_move`(*event*)

Callback for when the user changes the position of a plugin in the postprocessing order widget.

Updates the widget and writes the new order to the configuration.

Parameters **`event`** (`Tkinter.Event`) – Event from Tkinter

`create_driver_widgets`()
Create widgets for driver-related actions.

`create_plugin_widgets`()
Create widgets for plugin-related actions.

`load_values`()
Set widget state from configuration.

`set_orientation`(*target*)
Set target page on a device.

Prompts the user to connect a device, prompts to retry or cancel on failure. If successful, updates the target page setting on the device.

Parameters **`target`** (*unicode, one of "odd" or "even"*) – Target page to set on device

`configure_focus`()

Acquire auto-focus value from devices and update the configuration with it.

Prompts the user to connect a device, asks for cancel/retry on failure. On successful connection, acquires focus and writes the value to the configuration.

`save_config`()
Write configuration to disk.

`spreads.tkconfigure.`**`configure`**(*config*)
Initialize and display configuration dialog.

12.2 spreadsplug

spreadsplug package

This package contains all of the plugins and device drivers that are shipped with the application and supported by the spreads developers themselves.

In alphabetical order:

`spreadsplug.autorotate` Postprocessing plugin to rotate captured images according to their EXIF orientation tag.

`spreadsplug.dev.chdkcamera` Driver for Canon cameras with the CHDK firmware.

`spreadsplug.dev.gphoto2` Driver for cameras supported by libgphoto2

`spreadsplug.dev.dummy` Dummy driver that implements the driver interface and just spits out one of the two test images. Intended for rapid development, not for general usage.

`spreadsplug.djvubind` Output plugin to compress and bundle images (and OCRed text) into a single DJVU file using the *djvubind* utility.

`spreadsplug.gui` Subcommand plugin for a graphical wizard using Qt (via the PySide bindings)

`spreadsplug.hidtrigger` Trigger plugin to initiate a capture from USB HID devices (like foot-pedals or gamepads)

`spreadsplug.intervaltrigger` Trigger plugin to initiate a capture in a configurable interval.

`spreadsplug.pdfbeads` Output plugin to compress and bundle images (and OCRed text) into a single PDF file using the *pdfbeads* utility.

`spreadsplug.scantailor` Postprocesing plugin to put captured images through the ScanTailor application.

`spreadsplug.tesseract` Postprocessing plugin to perform optical character recognition on the images, using the *tesseract* application

`spreadsplug.web` Subcommand plugin for a RESTful HTTP API (implemented with Flask and Tornado) and a single-page JavaScript web application (implemented with ReactJS)

Trigger plugin that triggers in a configurable interval. Plugin to provide a RESTful HTTP API and a single-page web application for controlling the software.

The code for the plug in is split across the following server-side modules:

`spreadsplug.web.app` Contains the subcommand hook as well as the initialization code for the web application.

`spreadsplug.web.endpoints` WSGI endpoints that provide most parts of the RESTful interface, implemented with Flask.

`spreadsplug.web.handlers` Tornado HTTP handlers for long-polling and chunked downloading endpoints, as well as a WebSocket handler for sending out server-side events to all clients.

`spreadsplug.web.tasks` Implementations of long-running tasks that are performed in the background, across multiple request-response-cycles, through the Huey task queue.

`spreadsplug.web.discovery` Code for both advertising of postprocessing-servers via UDP multi-casting, as well as the auto-discovery of said servers from other instances.

`spreadsplug.web.util` Various utility classes and functions for the plugin.

`spreadsplug.web.winservice` Code for a simple Windows service that runs the application in the background and provides a small taskbar-icon to allow opening a browser and shutting down the appplication.

For the documentation of the client-side part, please refer to the following document: TODO

12.3 HTTP API

The web plugin also exposes all of its functions through a REST-ish API. You can use it to write small scripts or even for a full-blown Android or iPhone application, if you feel so inclined.

`GET /api/remote/templates`

Get option templates for all available plugins from a remote server.

Behaves exactly like *`GET /api/templates`*.

Query Parameters

- **`server`** – Hostname of remote server

Response Headers

- Content-Type[50] – *`application/json`*

`GET /api/remote/discover`

Get list of available postprocessing servers on network.

Response Headers

- Content-Type[51] – *`application/json`*

Request JSON Object

- **`servers`** (*array*) – List of available server addresses

`POST /api/system/shutdown`

Shut down device.

Requires that the user running the application has permission to run *shutdown -h now* via *sudo*. Note that this endpoint will never send a response, clients should take this into account and set a low timeout value.

`GET /api/remote/plugins`

Get available plugin names from a remote server, grouped by type.

Behaves exactly like *`GET /api/plugins`*.

Query Parameters

- **`server`** – Hostname of remote server

Response Headers

[50] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[51] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17

- Content-Type[52] – *application/json*

GET /api/remote/config

Get default configuration from a remote server.

Behaves exactly like *GET /api/config*.

Query Parameters

- **server** – Hostname of remote server

Response Headers

- Content-Type[53] – *application/json*

POST /api/system/reboot

Reboot device.

Requires that the user running the application has permission to run *shutdown -r now* via *sudo*. Note that this endpoint will never send a response, clients should take this into account and set a low timeout value.

GET /api/templates

For every activated plugin, get all option templates.

Response Headers

- Content-Type[54] – *application/json*

POST /api/workflow

Create a new workflow.

Request Headers

- Accept[55] – *application/json*

Request JSON Object

- **config** (*object*) – Configuration for new workflow
- **metadata** (*object*) – Metadata for new workflow

Response Headers

- Content-Type[56] – *application/json*

Status Codes

- 200 OK[57] – When everything was OK.
- 400 Bad Request[58] – When validation of configuration or metadata failed.

GET /api/workflow

Return a list of all workflows.

Response Headers

[52] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[53] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[54] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[55] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.1
[56] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[57] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2.1
[58] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.1

- Content-Type[59] – *application/json*

GET /api/plugins

Get names of available and activated postprocessing and output plugins.

Response Headers

- Content-Type[60] – *application/json*

Response JSON Object

- **postprocessing** (*array*) – List of postprocessing plugin names
- **output** (*array*) – List of output plugin names

GET /api/config

Get global default configuration.

Response Headers

- Content-Type[61] – *application/json*

PUT /api/config

Update global default configuration.

If *core* or *web* settings were modified, the application will be restarted.

Request Headers

- Content-Type[62] – *application/json*

Response Headers

- Content-Type[63] – *application/json*

POST /api/reset

Restart the application.

Note that this endpoint will never send a response, clients should take this into account and set a low timeout value.

GET /api/isbn

Search for ISBN records.

Query Parameters

- **q** – Search query

Response Headers

- Content-Type[64] – *application/json*

Response JSON Object

- **results** (*array*) – Matching ISBN records

Status Codes

[59]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[60]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[61]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[62]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[63]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[64]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17

- 200 OK[65] – When the query was successful
- 400 Bad Request[66] – When no search query was supplied

GET /api/log
Get application log.

Query Parameters

- **start** – Index of first message (default: *0*)
- **count** – Number of messages to return (default: *50*)
- **level** – Maximum log level to be included in messages (default: *INFO*)

Response Headers

- Content-Type[67] – `application/json`

Response JSON Object

- **total_num** (*boolean*) – Total number of messages
- **messages** (*array*) – Requested messages

GET /api/workflow/(**workflow:** *workflow*)**/page/**
int: *number***/***img_type***/***plugname***/thumb** Get thumbnail for a page image.

GET /api/workflow/(**workflow:** *workflow*)**/page/**
int: *number***/***img_type***/thumb** Get thumbnail for a page image.

POST /api/workflow/(**workflow:** *workflow*)**/page/**
int: *number***/***img_type***/crop** Crop a page image in place.

GET /api/workflow/(**workflow:** *workflow*)**/page/**
int: *number***/***img_type***/***plugname* Get image for requested page.

Parameters

- **workflow** (*str*) – UUID or slug for a workflow
- **number** (*int*) – Capture number of requested page
- **img_type** (str, one of *raw* or *processed*) – Type of image
- **plugname** (*str*) – Only applicable if *img_type* is *processed*, selects the desired processed file by its key in the `spreads.workflow.Workflow.processed_images` dictionary.

Query Parameters

- **width** – Optionally scale down image to the desired width
- **format** – Optionally convert image to desired format. If *browser* is specified, non-JPG or PNG images will be converted to PNG.

Response Headers

[65] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2.1
[66] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.1
[67] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17

- Content-Type[68] – Depends on value of *format*, by default the mime-type of the original image.

GET /api/workflow/(**workflow:** *workflow*)**/page/**
int: *number***/***img_type* Get image for requested page.

Parameters

- **`workflow`** (*str*) – UUID or slug for a workflow
- **`number`** (*int*) – Capture number of requested page
- **`img_type`** (str, one of *raw* or *processed*) – Type of image
- **`plugname`** (*str*) – Only applicable if *img_type* is *processed*, selects the desired processed file by its key in the `spreads.workflow.Workflow.processed_images` dictionary.

Query Parameters

- **`width`** – Optionally scale down image to the desired width
- **`format`** – Optionally convert image to desired format. If *browser* is specified, non-JPG or PNG images will be converted to PNG.

Response Headers

- Content-Type[69] – Depends on value of *format*, by default the mime-type of the original image.

GET /api/workflow/(**workflow:** *workflow*)**/output/**
fname Download an output file.

Parameters

- **`workflow`** (*str*) – UUID or slug for the workflow to download from
- **`fname`** (*str*) – Filename of the output file to download

Status Codes

- 200 OK[70] – Everything OK.
- 404 Not Found[71] – Workflow or filename not found

GET /api/workflow/(**workflow:** *workflow*)**/page/**
int: *number* Get a single page.

Parameters

- **`workflow`** (*str*) – UUID or slug for a workflow
- **`number`** (*int*) – Capture number of requested page

Response Headers

- Content-Type[72] – *`application/json`*

[68]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[69]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[70]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2.1
[71]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.5
[72]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17

`DELETE /api/workflow/`(workflow: *workflow*)`/page/`
int: *number* Remove a single page from a workflow.

`POST /api/workflow/`(workflow: *workflow*)`/prepare_capture`
Prepare capture for the requested workflow.

`POST /api/workflow/`(workflow: *workflow*)`/finish_capture`
Wrap up capture process on the requested workflow.

`GET /api/workflow/`(workflow: *workflow*)`/download`

Redirect to download endpoint (see `spreadsplug.web.handlers.ZipDownloadHandler` or `spreadsplug.web.handlers.TarDownloadHandler`) with proper filename set.

Parameters

- **`workflow`** (*str*) – UUID or slug for the workflow to download

Query Parameters

- **`fmt`** – Archive format for download (*zip* or *tar*, default: *tar*)

Status Codes

- 302 Found[73] – Redirects to *:http:get:'/api/workflow/\ (str:workflow_id)/download/\ (str:workflow_slug).(str:archive_extension)'*

`POST /api/workflow/`(workflow: *workflow*)`/transfer`
Enqueue workflow for transfer to an attached USB storage device.

Requires that the *python-dbus* package is installed.

Once the transfer was succesfully enqueued, watch for the `spreadsplug.web.tasks.on_transfer_started` which is emitted when the transfer actually started and subsequently `spreadsplug.web.tasks.on_transfer_progressed` and `spreadsplug.web.tasks.on_transfer_completed`.

Parameters

- **`workflow`** (*str*) – UUID or slug for the workflow to be transferred

Status Codes

- 200 OK[74] – When the transfer was successfully enqueued.
- 500 Internal Server Error[75] – When the *python-dbus* package was not found.
- 503 Service Unavailable[76] – When no removable USB device could be found for mounting.

`POST /api/workflow/`(workflow: *workflow*)`/capture`
Trigger a capture on the requested workflow.

Optional parameter 'retake' specifies if the last shot is to be retaken.

[73] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.3.3
[74] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2.1
[75] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1
[76] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.4

Returns the number of pages shot and a list of the pages captured by this call in JSON notation.

POST /api/workflow/(**workflow:** *workflow*)**/process**

Enqueue the specified workflow for postprocessing.

POST /api/workflow/(**workflow:** *workflow*)**/submit**

Enqueue workflow for submission to a postprocessing server.

It is possible to submit a configuration object that should be used on the remote end for the workflow. Optionally, it can be specified if postprocessing and output generation should immediately be enqueued on the remote server.

Once the submission was succesfully enqueued, watch for the `spreadsplug.web.tasks.on_submit_started` which is emitted when the submission actually started and subsequently `spreadsplug.web.tasks.on_submit_progressed`, `spreadsplug.web.tasks.on_submit_completed` and `spreadsplug.web.tasks.on_submit_error`.

Request Headers

- Accept[77] – *application/json*

Parameters

- **workflow** (*str*) – UUID or slug for the workflow to be submitted

Request JSON Object

- **server** (*string*) – Address of server to submit to
- **config** (*object*) – Configuration to use for workflow on remote server.
- **start_process** (*boolean*) – Whether to enqueue workflow for postprocessing on the remote server.
- **start_output** (*boolean*) – Whether to enqueue workflow for output generation on the remote server.

Status Codes

- 200 OK[78] – When the transfer was successfully enqueued.
- 400 Bad Request[79] – When no postprocessing server was specified
- 500 Internal Server Error[80] – When the *python-dbus* package was not found.
- 503 Service Unavailable[81] – When no removable USB device could be found for mounting.

POST /api/workflow/(**workflow:** *workflow*)**/output**

Enqueue the specified workflow for output generation.

[77]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.1
[78]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2.1
[79]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.1
[80]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1
[81]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.4

GET /api/workflow/(**workflow:** *workflow*)**/page**
Get all pages for a workflow.

Parameters

- **`workflow`** (*str*) – UUID or slug for a workflow

Response Headers

- Content-Type[82] – *application/json*

DELETE /api/workflow/(**workflow:** *workflow*)**/page**
Delete multiple pages from a workflow with one request.

GET /api/workflow/(**workflow:** *workflow*)
Return a single workflow.

Parameters

- **`workflow`** (*str*) – UUID or slug for a workflow

Response Headers

- Content-Type[83] – *application/json*

PUT /api/workflow/(**workflow:** *workflow*)
Update a single workflow.

Parameters

- **`workflow`** (*str*) – UUID or slug for the workflow to be updated

Request JSON Object

- **`config`** (*object*) – Updated workflow configuration
- **`metadata`** (*object*) – Updated workflow metadata

Response Headers

- Content-Type[84] – *application/json*

Status Codes

- 200 OK[85] – When everything was OK.
- 400 Bad Request[86] – When validation of configuration or metadata failed.

DELETE /api/workflow/(**workflow:** *workflow*)
Delete a single workflow from database and disk.

Parameters

- **`workflow`** (*str*) – UUID or slug for the workflow to be updated

Status Codes

- 200 OK[87] – When deletion was succesful

[82]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[83]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[84]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[85]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2.1
[86]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.1
[87]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2.1

GET /api/isbn/(*isbn*)

Get metadata for a given ISBN number.

Parameters

- **isbn** (str/unicode with valid ISBN-10 or ISBN-13, optionally prefixed with *isbn:*) – ISBN number to retrieve metadata for

Response Headers

- Content-Type[88] – *application/json*

Status Codes

- 200 OK[89] – When the ISBN was valid and a match was found.
- 400 Bad Request[90] – When the ISBN was invalid or no match was found.

GET /static/(**path:** *filename*)

Function used internally to send static files from the static folder to the browser.

New in version 0.5.

[88]http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17
[89]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.2.1
[90]http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.1

CHAPTER 13

Changelog

13.1 0.5 (2014/03/??)

- A web interface that currently supports creating workflows, capturing images and downloading them as a ZIP file.
- New plugins to trigger capture across all interfaces: 'hidtrigger' for USB HID devices, 'intervaltrigger' to trigger a capture in regular intervals
- Use new, optimized JPEG processing library
- Plugin API now useses mixin classes to declare which hooks are implemented
- Made 'chdkcamera' driver more resilient

13.2 0.4.2 (2014/01/05)

- Fix packaging issues
- Small bugfix for older Tesseract versions

13.3 0.4.1 (2013/12/25)

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

13.4 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

13.5 0.3.3 (2013/08/28)

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

13.6 0.3.2 (2013/08/24)

- Fixes a critical bug in the devices drivers

13.7 0.3.1 (2013/08/23)

- Fixes a bug that prevented spreads to be installed

13.8 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

13.9 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

13.10 0.1 (2013/06/23)

- Initial release