pyprika Documentation

Release 1.0.0

Paul Kilgo

February 16, 2014

Pyprika is a Python library for parsing and managing recipes. Its major features are:

- Support for recognizing a human-friendly representations of quantities and measurements normally seen in cook books.
- Parsing a custom YAML spec for storing recipes on disk.
- A simple command line utility `kit`, which makes use of the library for recipe management.

Contents:

CHAPTER 1

pyprika package

The pyprika package contains the primary API.

1.1 Input and output

pyprika.**load**(*fp*, *loader=None*, ***kw*)
Load fp, a file-like object

The file is assumed to be a pyprika-compliant YAML document. If the document contains a sequence, a list of Recipe objects will be returned. Otherwise, a single Recipe object should be returned.

Note that this function wraps the underlying exceptions thrown by Recipe.from_dict() under the assumption it is due to a malformed document, but the original traceback is preserved.

Parameters

- **fp** (*file-like*) – the file-like object containing the document to load
- **loader** (*callable*) – takes one positional argument and optional arguments and returns a dict (defaults to yaml.load)
- ****kw** – passed through to loader

Raises LoadError if there was an error in the loading of the document, usually indicative of a syntax error

Returns the recipe data contained in the stream

Return type Recipe or list of Recipe

pyprika.**loads**(*data*, *loader=None*, ***kw*)
Load recipe from string data.

This wraps data in a cString.StringIO and calls load() on it.

See load() for more information.

Parameters **data** (*str*) – recipe document data

Returns the recipe data contained in data

Return type Recipe or list of Recipe

pyprika.**dump**(*recipe*, *fp*, *dumper=None*, ***kw*)
Dump recipe to a file-like object

Parameters

- **recipe** (*Recipe*) – the recipe to dump
- **fp** (*file-like*) – the file stream to dump to
- **dumper** (*callable*) – a callable which takes two positional arguments, the first a dict and the second a file stream, and optional keyword arguments and encodes the recipe to the file stream (defaults to yaml.dump)
- ****kw** – passed through to dumper

pyprika.**dumps**(*recipe*, *dumper=None*, ***kw*)

Dump recipe object as a string.

This is a convenience method to dump to a StringIO object.

See `dump()` for parameter details.

Returns recipe encoded as a string

Return type str

1.2 API classes

1.2.1 Recipe

class pyprika.**Recipe**(*name*)

Class for representing a recipe.

Variables

- **name** (*str*) – human-friendly name of recipe
- **index** – optional application-specific indexing value
- **servings** – number of servings or a range (a 2-item tuple)
- **source** (*str*) – human-friendly source of recipe
- **source_url** (*str*) – URL source to for recipe
- **prep_time** (*Quantity*) – total preparation time for recipe
- **cook_time** (*Quantity*) – total cooking time for recipe
- **notes** (*str*) – miscellaneous data about recipe
- **ingredients** (*list*) – list of Ingredient objects
- **directions** (*list*) – list of instructions to prepare recipe

classmethod **from_dict**(*d*)

Creates a new recipe from a dictionary.

Parameters **d** (*dict*) – the dictionary to convert

Raises

- **FieldError** – if a field is missing, invalid, or not well-formed
- **ParseError** – if a Pyprika syntax error is present

Returns the resulting recipe

Return type Recipe

to_dict (*serialize=False*)
Return a dictionary representing the Recipe.

Parameters **serialize** (*bool*) – convert as much as possible to primitive types

Returns a dictionary mapping attribute names to values

Return type dict

1.2.2 Ingredient

class pyprika.**Ingredient** (*label*, *quantity=None*)
Class for representing ingredients.

Variables

- **label** (*str*) – the label of the ingredient (like `egg`)
- **quantity** (*Quantity*) – the amount of the ingredient, if any

classmethod **parse** (*s*)
Parse an object from a string. Valid strings are of the form:

```
[(quantity) ]label
```

Where `quantity` must be valid syntax to `Quantity.parse()` and `label` is any text not beginning with a value enclosed in parenthesis.

Parameters **s** (*str*) – string to parse

Raises **ParseError** on invalid syntax

Returns the resulting Ingredient

Return type Ingredient

1.2.3 Quantity

class pyprika.**Quantity** (*amount=None*, *unit=None*)
Class for representing quantity.

Quantities can either be a measurement (like `1 cup`) or a dimensionless amount (like `12`).

Variables

- **amount** – numeric amount of the quantity
- **unit** (*str*) – unit of the amount, if any

classmethod **parse** (*s*)
Parse an object from a string. Valid strings are of the form:

```
amount[ unit]
```

Where `unit` is unconstrained and `amount` is one of the following:

•an integer, like `4`

•a decimal number, like `4.5` (*not* scientific notation)

•a fraction, like `1/2`

•a mixed number, like `1 1/2`

Parameters **s** (*str*) – string to parse

Raises ParseError on invalid syntax

Returns the resulting Quantity

Return type Quantity

1.3 Exceptions

1.3.1 FieldError

class pyprika.**FieldError**
Raised when a constraint on a field is not met.

1.3.2 ParseError

class pyprika.**ParseError**
Raised on invalid syntax.

1.3.3 LoadError

class pyprika.**LoadError**(**args*, ***kwargs*)
A blanket exception to catch if there was an error loading a recipe.

Variables **cause** – the original exception, if any

CHAPTER 2

The `kit` utility

`kit` is a simple command line application of the pyprika library. It's meant for recipe management via the command line. It does require a little setup so that it knows where to find your recipes.

The name is somewhat a play on "kitchen" and "Git".

2.1 The `~/.kitrc` file

Your `.kitrc` governs the behavior of `kit`. Upon startup, `kit` searches the paths defined in this configuration file for recipes as well as the current directory.

For example, if this is in `~/.kitrc`:

```
paths:
  - /home/paul/recipes
  - /usr/local/share/recipes
```

Then it will search the paths `/home/paul/recipes` and `/usr/local/share/recipes` upon startup. The default behavior is to do a shallow search. If you want it to do a recursive search:

```
recursive: True
paths:
  - /home/paul/recipes
  - /usr/local/share/recipes
```

That's the basics. The reference should be enough.

2.2 Commands

`kit` is organized into subcommands, similar to a lot of other popular utilities you're used to using.

Since it's often inconsistent to refer to recipes by their name, `kit` indexes each recipe by taking the MD5 hash of the source file's contents. This has its obvious flaws (the index changes when the file contents does), so this is only done if a index has not been manually assigned in the source file.

2.2.1 Edit

Edits a recipe, using the first editor found in the environment variables `KIT_EDITOR`, `EDITOR`, and falling back on `pico`.

Usage:

```
kit edit index
```

2.2.2 List

Lists recipes in the registry by their index and name.

Usage:

```
kit ls
```

2.2.3 Show

Pretty-print a recipe to the command line. The recipe can optionally be scaled.

Usage:

```
kit show [-s|--scale SCALE] index
```

2.2.4 Search

Search for a recipe by terms in the title. The search can be case-insensitive when the `-i` flag is specified.

```
kit search [-i] search-term
```

2.2.5 Validate

Validate one or more input Pyprika recipes to verify it is correctly formed.

```
kit validate filename [filename...]
```

2.2.6 Which

Print the path to a recipe given its index.

```
kit which index
```

CHAPTER 3

Pyprika's YAML specification

Pyprika's YAML format was developed from an export syntax used by the Paprika smartphone application with some modifications primarily to make the syntax more rigid.

The following are the design goals of the syntax in order of importance:

- As little markup as possible
- Support the "natural" way of expressing cooking concepts
- Easily machine-processible where possible

This documentation will work from an example which utilizes all of the features to touch on:

```
name: Dumplings

index: DMP0001
servings: [4, 6]
source: The Internet
source_url: http://www.example.com/
prep_time: 5 min
cook_time: 45 min
notes: |
  Piping hot!

ingredients:
  - (1/2 tsp) baking powder
  - (1 1/2 cup) flour
  - salt
  - pepper
  - (2 tbsp) olive oil
  - (4.5 cups) water

directions:
  - Put it in a bowl.
  - Mix 'em up.
  - Bake.
```

Though it looks verbose, the only required field is `name`. The rest can safely be left as their defaults.

3.1 Field descriptions

name The name of the recipe.

index An indexing field which is not used internally by the library, but can be used by applications to refer to the recipe.

servings The number of servings the recipe produces. This may be an integer, or can be a 2-item list to represent a range (e.g. 2 servings or 2-4 servings).

source An unrestricted field for noting the source of the recipe.

source_url An unrestricted field for noting the URL from which the recipe originates. No validation is performed to ensure this is a URL.

prep_time The total time of preparation for the recipe. Must conform to quantity syntax.

cook_time The amount of cooking time for the recipe. Must conform to quantity syntax.

notes An unconstrained field for providing miscellaneous notes.

ingredients A list of ingredients needed for the recipe. Each item in the list must conform to ingredient syntax.

directions A list of directions (in order) for preparing the recipe. The items of the list are unconstrained.

3.2 Ingredient syntax

Ingredient syntax has two primary forms: unquantified and quantified. The unquantified form simply means the ingredient does not have an associated quantity, whereas the quantified form does.

3.2.1 Unquantified form

The unquantified form is the easiest to understand. The only restriction is the start of the string cannot contain parenthesis. So all of the following are examples of the unquantified form:

- `salt`
- `water (optional)`
- `black pepper`

3.2.2 Quantified form

The quantified form has a quantity or measurement associated with it. They look the same as the unquantified form, but with a measurement in parenthesis on the left. The following are all examples of the quantified form:

- `(1 cup) water`
- `(1 1/2 cup) unbleached flour`
- `(1) apple (optional)`

The text in the leading set of parantheses must follow the quantity syntax.

3.3 Quantity syntax

The quantity syntax is a means of expressing amounts, whether it be dimensionless (like a count) or dimensioned (like `1 cup`). In general, the quantity syntax follows this form:

```
number [unit]
```

Where `number` is one of the following:

- a non-negative integer (e.g. `0`, `12`)
- a decimal point number (e.g. `1.5`, `2.75`)
- a fraction (e.g. `1/2`, `3/4`)
- a mixed number (e.g. `1 1/2`, `2 3/4`)

Scientific notation is not supported for decimal point numbers.

`unit` is optional, and any text found after the number is considered a part of the unit. As an example, for `1 1/2 fl oz`, then `1 1/2` would be interpreted as the number and `fl oz` would be interpreted as the unit.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

p