Brewmeister Documentation

Release 0.1.0dev

Matthias Vogelgesang

August 07, 2014

Contents

Brewmeister is a server application to organize and control beer brewing processes. It provides an HTML interface for human brewers and a REST-API for machine consumption. It also works perfectly nice with its Brewslave minion.

CHAPTER 1

Features

- Based on Flask + MongoDB
- Simple REST API
- i18n and l10n for German and Czech
- Client- and server-side validation via JSON schemas
- Temperature control based on a state machine
- Bottle cap label generator
- Absolutely *no* security measures

CHAPTER 2

Documentation

Is hosted at readthedocs.org.

CHAPTER 3

Screenshot

Contents

4.1 Installation

Prepare a virtualenv and install all requirements:

```
$ pip install --upgrade -r requirements.txt
```

Setup a MongoDB instance, e.g.

```
$ sudo apt-get install mongodb-server
```

For testing purposes you can pre-populate the database with:

```
$ make init
```

Generate translation data base and run the debug server with:

```
$ make
```

By default, a dummy controller is running with which you can brew a virtual beer.

4.1.1 Customization

You can edit `brew/settings.py` and change the following configuration options:

`BREW_CONTROLLER_TYPE`	Can be either `dummy` or `arduino`.
`BREW_CONTROLLER_ARDUINO_DEVICE`	Device filename of the serial connection to the Arduino device. It is is `/dev/ttyUSB0` by default.
`BREW_CONTROLLER_DUMMY_SLOPE`	Temperature increase in degrees per minute of the dummy controller.

4.2 Development

4.2.1 Contributing

Brewmeister is free and open source software and you are encouraged to report bugs, contribute features and bug fixes as well as translating the Brewmeister into your language.

Bug reports and feature requests

All bugs and feature requests should be reported at the GitHub issue tracker.

Code contributions

Common open source practices apply to the Brewmeister development too. First of all, all code contributions are reviewed and merged through a GitHub pull request. Please base your changes on a feature branched off of master and not master itself. Name it according to your intended changes, e.g. `fix-bug-123` or `add-magic-hops`.

Within your code, you should follow PEP8 with one exception: the line length can be up to 100 characters per line instead of 80.

Translations

The easiest way to add or improve translations is to go to the Transifex project page and request a new language or start digging on the existing ones. This is the preferred way for translators, as the messages source file is uploaded when necessary.

You can also add and translate manually. First create a new language with `make createpo`, enter the targetted language code and edit the translation file in `brew/translations/<lang>/LC_MESSAGES/messages.po`. Once finished, you can add and commit this file and issue a pull request on GitHub.

4.2.2 Arduino Brew Control Protocol

The Arduino Brew Control Protocol (short ABCP) is a *simple*, *compact*, *stateless* and *command-based* wire protocol for communicating with a Brewmeister-compatible Arduino.

The following specification assumes, the *host* to be the machine that communicates with the Arduino via a serial line interface.

Protocol sequence

Communication is *always* initiated by the host using a simple call-response sequence:

1. Host sends command packet specifying either to *read* or to *write* data.
2. Host sends device packet specifying which device is addressed.
3. In case of a *write* command, the host sends the data.
4. The Arduino answers with a status code and depending on the command, optional data.

Command packet

The command packet is sent by the host and consists of one header byte

Code	Meaning
`0xf0`	Read data
`0xf1`	Write data

and one device byte.

Code	Instrument	Data type	Meaning
`0xf1`	Temperature	float	Temperature in degree Celsius.
`0xf2`	Heat	bool	On or off.
`0xf3`	Stir	bool	On or off.

Data types

Data can – as of now – be sent and read as floats or boolean data types. A float is a four byte IEEE compliant float data type in x86-compatible little endian format. The boolean type is one byte, with 0 denoting *false* and 1 denoting *true*.

4.2.3 RESTful HTTP API

Recipes

POST /api/recipe
Create a new recipe. The data must be encoded as a JSON data structure according to the JSON schema stored in `data/recipe.schema.json`.

PUT /api/recipe/ (**int:** *recipe_id*)
Recipe data of (*recipe_id*).

Brews

GET /api/brews
List of brew IDs.

GET /api/brews/ (**int:** *brew_id*) **/temperature**
Archived temperature data for (*brew_id*).

GET /api/brews/ (**int:** *brew_id*) **/label**
Return a PDF called `qr.pdf` containing small QR codes for bottle caps.

GET /api/brews/ (**int:** *brew_id*) **/temperature**
Get all recorded temperatures of the specified brew.

GET /api/brews/ (**int:** *brew_id*) **/label/prepare**
Prepare a label asynchronously.

PUT /api/brews/ (**int:** *brew_id*) **/note**
Update notes of brew.

GET /api/status
Status of the current brew.

Hardware access

PUT /api/reconnect
Try to reconnect again with the set controller.

GET /api/status/ (**str:** *device*)
Get running status of device.

PUT /api/start/ (**str:** *device*)
Start the device.

PUT /api/stop/ (**str:** *device*)
Stop the device.

CHAPTER 5

License

Brewmeister is created by Matthias Vogelgesang.

The favicon is an adapted beer icon from GLYPHICONS.com, released under Creative Commons Attribution 3.0 Unported (CC BY 3.0).

/api