FFFlash Documentation

Release 0.9a8

Frieder Griesshammer

December 15, 2015

Contents

Welcome to the FFFlash Documentation

Some random Links:

GitHub https://github.com/spookey/fffflash

PyPy https://pypi.python.org/pypi/fffflash

Read the Docs https://fffflash.readthedocs.org

This Program helps to manage Freifunk APIfiles.

To use it you need an existing APIfile. Create one here if necessary.

CHAPTER 1

Commandline

There is no configuration.

Everything **ffflash** needs to know is passed each time as shell parameters. (It's not much).

- To get help, use `ffflash.py -h`:

```
usage: ffflash [-h] [-s SIDECARS [SIDECARS ...]] [-n NODELIST] [-r RANKFILE]
               [-rc RANKCLIENTS] [-rf RANKOFFLINE] [-rn RANKONLINE]
               [-rp RANKPOSITION] [-rw RANKWELCOME] [-d] [-v]
               APIfile
```

- You always need to pass the location to your `APIfile`.
- `--nodelist` or `-n` expects the full url of your `nodelist.json` file, generated by the ffmap-backend.
- Do some number crunching and store the *scores* in the `--rankfile`. (Only works if passed together with an `--nodelist`).
  - `--rankwelcome` sets the initial *score* to start with.
  - `--rankposition` increases *score* by that value if any location data is provided.
  - `--rankonline` increases *score* by that value if node is online.
  - `--rankoffline` decreases *score* by that value if node is offline.
  - `--rankclients` increases *score* per client by that value.
- Pass one or more `--sidecars` (or `-s`) to merge content from there into the APIfile.
- Use `--dry` (or `-d`) to preview all changes. Then, nothing is written to the APIfile!
- If both `-s` and `-n` are omitted, no action is taken. So to just display your APIfile you could use this:

```
ffflash.py /path/to/your/ffapi_file.json -d
```

  (makes no sense anyway, better use `cat`)

- The `--verbose` (or `-v`) switch just displays more information.

CHAPTER 2

Documentation

2.1 Main

class ffflash.main.**FFFlash**(*args*)

This is the main object, which stores all relevant information, and the *ffflash.lib.api.FFApi* itself.

Parameters **args** – Namespace object from *ffflash.lib.args.parsed_args()*

access_for(*name*)

Check if it is save to access the api and/or depending files.

Parameters **name** – String specifier of part to access.

Returns True or False

load_api()

Populate api with *ffflash.lib.api.FFApi* with content loaded from location.

api is populated only once, this prevents accidental reloads.

log(*message*, *level=True*)

Very advanced Logger. For professional use only.

Parameters

- **message** – Some message string to display
- **level** – Severity of message. Can be anything as this is also the return value. There are three predefined rules:
  - True: *info* (is dismissed unless --verbose was given)
  - None: *warn*
  - False: *error*

Returns level

save()

Save content from api (*ffflash.lib.api.FFApi*) into location. A ffflash.lib.api.FFApi.timestamp() is triggered before saving.

set_timestamp()

Inject *ffflash.lib.clock.get_iso_timestamp()* into state.lastchange.

ffflash.main.**run**(*argv=None*)

Main function of FFFlash.

2.1.1 Info

To share common values like the package name or release string between setuptools, sphinx and the code itself, the info module is used.

class `ffflash.info.`**Info**
Shared Information is stored in a class, for easy access.

rst_epilog

Return str Restructured Text with substitutions for the info values, so they can be displayed in sphinx documentation

2.2 Nodelist

If a `--nodelist` is passed, it will be used to count online Nodes and Clients from there. It can either be a local file, or if this does not exist, it is interpreted as URL, and fetched from there.

If successful, the numbers will be added to the APIfile:

```
{
  "state": {
    "nodes": 0,
    "description": ""
  }
}
```

Would be changed to this:

```
{
  "state": {
    "nodes": 23,
    "description": "[23 Nodes, 42 Clients]"
  }
}
```

`ffflash.inc.nodelist.`**_nodelist_count**(*ff*, *nodelist*)
Count online nodes and sum up their clients from a nodelist.

Parameters

- **ff** – running *ffflash.main.FFFlash* instance
- **nodelist** – nodelist from *_nodelist_fetch()*, should contain a list of dictionaries at the key *nodes*

Returns Tuple of counted nodes and clients

`ffflash.inc.nodelist.`**_nodelist_dump**(*ff*, *nodes*, *clients*)
Store the counted numbers in the api-file.

Sets the key `state.nodes` with the node number.

Leaves `state.description` untouched, if any already present. If empty, or the pattern `\[[\d]+ Nodes, [\d]+ Clients\]` is matched, the numbers in the pattern will be replaced.

Parameters

- **ff** – running *ffflash.main.FFFlash* instance
- **nodes** – Number of online nodes

- **clients** – Number of their clients

Returns `True` if `api` was modified else `False`

`fflash.inc.nodelist.`**_nodelist_fetch**(*ff*)
Determines if `--nodelist` was a file or a url, and tries to fetch it. Validates nodelist to be json and to have the *version*, *nodes* and *updated_at* keys.

Parameters **ff** – running *fflash.main.FFFlash* instance

Returns the unpickled nodelist or `False`/`None` on error

`fflash.inc.nodelist.`**handle_nodelist**(*ff*)
Entry function to receive a `--nodelist` and store determined results into both `api` and `--rankfile` (if specified).

Parameters **ff** – running *fflash.main.FFFlash* instance

Returns `True` if `api` was modified else `False`

2.2.1 Rankfile

`fflash.inc.rankfile.`**_rankfile_dump**(*ff*, *rankfile*, *ranks*)
Store ranks in `rankfile`. Also sets a timestamp and writes the release string into the output.

Parameters

- **ff** – running *fflash.main.FFFlash* instance
- **rankfile** – validated path to the `rankfile`
- **ranks** – content to store

Returns `True` on success, or `False` on error

`fflash.inc.rankfile.`**_rankfile_load**(*ff*)
Load either existing `rankfile` from disk, or create empty stub if one does not exist yet. Path and extension (*json*) get validated.

Parameters **ff** – running *fflash.main.FFFlash* instance

Returns

Tuple of either (`False`, `None`) on error or:

- validated path to the `rankfile`
- `rankfile` content

`fflash.inc.rankfile.`**_rankfile_score**(*ff*, *ranks*, *nodelist*)
Do some number crunching.

Parameters

- **ff** – running *fflash.main.FFFlash* instance
- **ranks** – rankfile content from *_rankfile_load()*, should contain a list of dictionaries at the key *nodes*
- **nodelist** – nodelist from *fflash.inc.nodelist._nodelist_fetch()*, should contain a list of dictionaries at the key *nodes*

Returns `ranks` with new scores, sorted by score

`fflash.inc.rankfile.`**handle_rankfile**(*ff*, *nodelist*)
Entry function gather results from a retrieved `--nodelist` to store it into the `--rankfile`.

Parameters **ff** – running *ffflash.main.FFFlash* instance

Returns `True` if rankfile was modified else `False`

2.3 Sidecars

Use one ore more Sidecars to merge content from there into the APIfile.

Sidecars are either *yaml* or *json* files. This is determined by the extension in the filename.

The filename is a dot-separated path into the keys of the APIfile. If that path does not exist in the APIfile, it will be ignored.

Only if a Sidecar itself does not exist yet or is empty, it will be generated with the contents read from the APIfile.

Assuming this APIfile:

```
{
  "support": {
    "club": {
      "name": "Supporter Club",
      "city": "Generic City",
      "street": "Some Street 23",
      "zip": "23425"
    },
    "donations": {
      "bankaccount": {
        "BIC": "ABC123DEFXX",
        "IBAN": "GC13370000000123456789",
        "usage": "I like cash"
      }
    }
  }
}
```

Valid filenames and their content would be these:

- `/path/to/your/sidecars/support.club.city.yaml`:

```
Generic City
...
```

- `/path/to/your/sidecars/support.club.yaml`:

```
city: Generic City
name: Supporter Club
street: Some Street 23
zip: '23425'
```

- `/path/to/your/sidecars/support.yaml`:

```
club:
    city: Generic City
    name: Supporter Club
    street: Some Street 23
    zip: '23425'
donations:
    bankaccount:
        BIC: ABC123DEFXX
```

```
            IBAN: GC13370000000123456789
            usage: I like cash
```

- /path/to/your/sidecars/support.donations.bankaccount.usage.json:

```
"I like cash"
```

- /path/to/your/sidecars/support.donations.bankaccount.json:

```
{
  "BIC": "ABC123DEFXX",
  "IBAN": "GC13370000000123456789",
  "usage": "I like cash"
}
```

- /path/to/your/sidecars/support.donations.json:

```
{
  "bankaccount": {
    "BIC": "ABC123DEFXX",
    "IBAN": "GC13370000000123456789",
    "usage": "I like cash"
  }
}
```

Invalid filenames would be these:

- /path/to/your/sidecars/support.club.city.txt:

  Wrong extension

- /path/to/your/sidecars/support.industry.json:

  Key *industry* is not present in APIfile.

- /path/to/your/sidecars/support.donations.bankaccount.iban.yaml:

  iban can't be found, it's case sensitive. Use *IBAN* instead.

Duplicated Sidecar content is handled like this. Assuming these Sidecars with this content:

- support.club.street.yaml:

```
Same Street 5
...
```

- support.club.yaml:

```
city: Generic City
name: Supporter Club
street: Another Street 42
zip: '23425'
```

The List of Sidecars is sorted, so the longer filename is handled first. So the shorter filename wins, the result is then:

```
{
  "support": {
    "club": {
      "name": "Supporter Club",
      "city": "Generic City",
      "street": "Another Street 42",
      "zip": "23425"
```

```
    }
}
```

ffflash.inc.sidecars.**_sidecar_dump**(*ff*, *sidepath*, *content*, *fields*, *as_yaml*)
Stores `content` both in `api` and `sidepath`.

Parameters

- **ff** – running *ffflash.main.FFFlash* instance
- **sidepath** – full path to the sidecar
- **content** – the value to store into sidecar/api-file
- **fields** – key-names into api-file
- **as_yaml** – dump as *yaml* instead of *json*

Returns `True` if `sidepath` was modified else `False`

ffflash.inc.sidecars.**_sidecar_load**(*ff*, *sidepath*, *fields*, *as_yaml*)
Loads content from `sidepath` if it exists, otherwise returns the values from the `api` instead. This is only done, if `fields` exist in `api`.

Parameters

- **ff** – running *ffflash.main.FFFlash* instance
- **sidepath** – full path to the sidecar
- **fields** – key-names into api-file
- **as_yaml** – load as *yaml* instead of *json*

Returns The loaded content of `sidepath` or `False`/`None` on error

ffflash.inc.sidecars.**_sidecar_path**(*ff*, *sc*)
Check passed sidecars for valid paths, format (*json* or *yaml*) and for valid filenames (no double dots).

Parameters

- **ff** – running *ffflash.main.FFFlash* instance
- **sc** – sidecar as passed by user

Returns

Tuple of either (`False`, `None`, `None`) on error or:

- normalized and full path to `sc`
- unvalidated key-names into api-file
- `True` if `sc` is a *yaml* file, `False` if it's *json*

ffflash.inc.sidecars.**handle_sidecars**(*ff*)
Entry function to handle passed `--sidecars`. Validating locations, names and content of sidecars. Generating them if necessary and update `api`.

Parameters **ff** – running *ffflash.main.FFFlash* instance

Returns `True` if any sidecar was modified else `False`

2.4 Library

2.4.1 Api

class ffflash.lib.api.**FFApi**(*content*)

Helper class provide some easy way to access and modify dictionaries. It only provides reading and replacing already existing keys.

Parameters **content** – The initial data to work with

pretty()

Return str current content in a human readable way using **pprint.pformat**

pull(**fields*)

Retrieve contents from deep down somewhere in the dictionary.

Parameters **fields** – one or more key names to retrieve

push(*value*, **fields*)

Replace contents deeply inside the dictionary, if the key already exists.

Parameters

- **value** – the actual data to be written
- **fields** – one or more key names where to write value

2.4.2 Args

ffflash.lib.args.**parsed_args**(*argv=None*)

Parse arguments from commandline

Parameters **argv** – List of Arguments to parse. - If omitted **sys.argv** is used

Return Namespace arguments from **ArgumentParser** for argv

2.4.3 Clock

ffflash.lib.clock.**get_iso_timestamp**(*dt=None*)

Generate iso timestrings

Parameters **dt** – custom datetime object, or now() if None

Return str iso representation of dt

2.4.4 Files

ffflash.lib.files.**dump_file**(*location*, *content*, *as_yaml=False*)

Pickle either *json* or *yaml* into a file

Parameters

- **location** – path where to pickle into
- **content** – data to store
- **as_yaml** – output as *yaml* instead of *json*

ffflash.lib.files.**load_file**(*location*, *fallback=None*, *as_yaml=False*)
Unpickle either *json* or *yaml* from a file

Parameters

- **location** – path where to unpickle from
- **fallback** – data to return in case of unpickle failure
- **as_yaml** – read as *yaml* instead of *json*

Returns unpickled data from `location`

ffflash.lib.files.**read_file**(*location*, *fallback=None*)
Read string data from files

Parameters

- **location** – filename where to write to
- **fallback** – data to return in case of read failure

Returns read data from `location` if successful else `fallback`

ffflash.lib.files.**write_file**(*location*, *data*)
Write string data into files

Parameters

- **location** – filename where to write to
- **data** – content to write into `filename`

Returns `data` if successful

2.4.5 Locations

ffflash.lib.locations.**check_file_extension**(*location*, **extensions*)
Validate path for a selection of extensions.

Parameters

- **location** – path to check
- **extensions** – one or more extensions the `location` should end with

Return tuple (basename of `location`, extension of `location`) or (`None`, `None`) if extension did not match

ffflash.lib.locations.**check_file_location**(*location*, *must_exist=False*)
Validate path for a file.

Checks for the parent folder to exist, and that `location` itself is not a folder. Optionally, if `location` is an already existing file.

Parameters

- **location** – path to check
- **must_exist** – check also if `location` really exists and is a file

Return str validated path of `location` if all above conditions are met or `None`

ffflash.lib.locations.**get_basedir**()
Fancy helper to find project's basedir. Use *locate_file()* to reach into the package folder.

Returns full absolute path to ffflash's basedir

ffflash.lib.locations.**locate_file**(**parts*, *must_exist=False*)
Find files inside :meth:`get_basedir`.

Parameters

- **parts** – trail to your file. e.g. `bla/fasel/blubb` would be `'bla', 'fasel', 'blubb'`
- **must_exist** – check if located file really exists and is a file see `ffflash.lib.files.check_file_location()` for more

Returns full absolute path to desired file, or `None` on error

2.4.6 Remote

ffflash.lib.remote.**fetch_www**(*url*, *fallback=None*, *timeout=5*)
Contextmanager to retrieve content from the web

Parameters

- **url** – URL to fetch
- **fallback** – what to return instead in case of error
- **timeout** – timeout to pass to `urllib.request`

Yield str fetched result as unicode string, or `fallback`

ffflash.lib.remote.**fetch_www_struct**(*url*, *fallback=None*, *timeout=5*, *as_yaml=False*)
Helper to unpickle either *json* or *yaml* from fetched files

Parameters

- **url** – URL to fetch
- **fallback** – what to return in case of (fetch or unpickle) error
- **timeout** – timeout to pass down to *`fetch_www()`*
- **as_yaml** – load content as *yaml* instead of *json*

Returns unpickled data from `url`

2.4.7 Struct

ffflash.lib.struct.**dump_struct**(*content*, *as_yaml=False*)
Contextmanager to pickle either *json* or *yaml* into a string

Parameters

- **content** – data to pickle
- **as_yaml** – output as *yaml* instead of *json*

Yield str pickled `content`

ffflash.lib.struct.**load_struct**(*content*, *fallback=None*, *as_yaml=False*)
Contextmanager to unpickle either *json* or *yaml* from a string

Parameters

- **content** – string to unpickle
- **fallback** – data to return in case of unpickle failure

- **as_yaml** – read as *yaml* instead of *json*

Yield unpickled `content`

`fffflash.lib.struct.`**merge_dicts**(*first*, *second*)
Merge nested dictionaries deeply

Parameters

- **first** – Source dictionary
- **second** – Dictionary to merge into `first`

Return dict merged dictionaries

2.4.8 Text

`fffflash.lib.text.`**make_pretty**(*data*)

Parameters **data** – ugly data

Return str pretty data formatted using **pprint.pformat**, or `None` if data was too ugly

`fffflash.lib.text.`**replace_text**(*rx*, *replacement*, *text*)
Replace text if `rx` matches.

Parameters

- **rx** – regex to match on `text`
- **replacement** – content to put into `text` on `rx` match
- **text** – content to work on

Return str `text` with replaced parts, or unchanged `text`

`fffflash.lib.text.`**search_text**(*rx*, *text*)
Safe search `text` with regex.

Parameters

- **rx** – regex to match on `text`
- **text** – content to work on

Returns either `None` if `rx` is not in `text` or **match-object** of **re**.

2.5 Setup

FFFlash is available as package, you can find the newest version here:

pypy https://pypi.python.org/pypi/fffflash

Most requirements are not necessary for normal operations, only for developing. The most notable exception is **PyYAML**.

```
pytest == 2.7.3
pytest-cov == 2.1.0
pytest-runner == 2.6.2
python_dateutil == 2.4.2
PyYAML == 3.11
setuptools == 18.3.1
```

```
Sphinx == 1.3.1
watchdog == 0.8.3
```

find_requirements() figures out what dependencies are required.

To install/update latest version of ffflash:

```
sudo pip3 install -U ffflash
```

To install all requirements from a local clone for developing:

```
sudo pip3 install -U -r requirements.txt
```

setup.**find_requirements**(**names*)

Parameters **names** – one or more required package names

Returns list package lines from `requirements.txt` whose lowercased name is in `names`

setup.**local_file**(*name*)

Parameters **name** – filename to read relative from current directory

Returns str content of `name` or empty string on failure

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

f

s

Symbols

A

C

D

F

G

H

I

L

M

P

R

S

W