RetroArcher

ReenigneArcher

Feb 10, 2024

ABOUT

1 Overview **3**
- 1.1 About . . . 3
- 1.2 Integrations . . . 3
- 1.3 Support . . . 3
- 1.4 Downloads . . . 3
- 1.5 Stats . . . 4
- 1.6 Project Goals . . . 4

2 Installation **5**
- 2.1 Binaries . . . 5
- 2.2 Docker . . . 5
- 2.3 Source . . . 5

3 Usage **7**
- 3.1 Network . . . 7
- 3.2 Arguments . . . 7

4 Docker **9**
- 4.1 lizardbyte/retroarcher . . . 9

5 Build **11**
- 5.1 Clone . . . 11
- 5.2 Setup venv . . . 11
- 5.3 Install Python Requirements . . . 11
- 5.4 Compile Locales . . . 12
- 5.5 Install NPM Requirements . . . 12
- 5.6 Compile Docs . . . 12
- 5.7 Compile Binary . . . 12
- 5.8 Remote Build . . . 12

6 Changelog **13**

7 Contributing **15**

8 Localization **17**
- 8.1 CrowdIn . . . 17
- 8.2 Extraction . . . 18

9 Testing **19**
- 9.1 Flake8 . . . 19
- 9.2 Sphinx . . . 19

9.3 pytest 19

10 retroarcher **21**

11 pyra.__init__ **23**

12 pyra.config **25**

13 pyra.definitions **29**

14 pyra.hardware **33**

15 pyra.helpers **37**

16 pyra.locales **43**

17 pyra.logger **45**

18 pyra.threads **55**
18.1 Routine Listings 55
18.2 Examples 55

19 pyra.tray_icon **57**

20 pyra.webapp **63**

Python Module Index **69**

Index **71**

LizardByte has this documentation hosted on Read the Docs.

CHAPTER

ONE

OVERVIEW

1.1 About

RetroArcher is a game streaming server application. This project is under development and nowhere near ready for use.

1.2 Integrations

1.2.1 CrowdIn Localization

1.3 Support

Our support methods are listed in our LizardByte Docs.

1.4 Downloads

1.5 Stats

1.6 Project Goals

Plugin Framework
Develop a framework friendly to plugins that allows easy expansion of the application.

Plugin types may be:

- Game Stream Hosts (Sunshine)
- Clients (Android, AppleTV, iOS, PC [Linux, MacOS, Windows], Xbox One/Series S/X, etc.)
- Emulators (Cemu, RetroArch, RPCS3, etc.)
- Game Stores (Epic Games, Steam, Microsoft Store, etc.)
- Consoles (game streaming Xbox One/Series S/X, PS4/5, etc.)
- Media Servers (Emby, Jellyfin, Kodi, Plex, etc.)
- Misc. (anything that doesn't fit a category above)

Replace Existing RetroArcher.bundle
RetroArcher.bundle has been renamed to RetroArcher-plex and it has significantly changed:

- No longer responsible for scanning games
- No longer responsible for connecting to clients
- No longer responsible for starting games
- No longer gets metadata from IGDB
  - Metadata is now collected from our own db based on IGDB.
  - In the future metadata will be collected by this server application, and the Plex plugin will make an API request to this server to get the metadata.

CHAPTER

TWO

INSTALLATION

The recommended method for running RetroArcher is to use the *binaries* bundled with the latest release.

2.1 Binaries

Binaries of RetroArcher are created for each release. They are available for Linux, MacOS, and Windows. Binaries can be found in the latest release.

2.2 Docker

Docker images are available on Dockerhub.io and ghcr.io.

See *Docker* for additional information.

2.3 Source

Caution: Installing from source is not recommended most users.

1. Follow the steps in *Build* except for *Compile Binary*.
2. Run the following within your activated venv.

```
python retroarcher.py
```

CHAPTER

THREE

USAGE

3.1 Network

RetroArcher will be available on port 9696 by default.

3.2 Arguments

To get a list of available arguments run the following:

Binary

```
retroarcher --help
```

Python

```
python retroarcher.py --help
```

CHAPTER

FOUR

DOCKER

4.1 lizardbyte/retroarcher

4.1.1 Using docker run

Create and run the container (substitute your <values>):

```
docker run -d \
  --name=retroarcher \
  --restart=unless-stopped
  -v <path to data>:/config \
  -e PUID=<uid> \
  -e PGID=<gid> \
  -e TZ=<timezone> \
  -p 9696:9696 \
  lizardbyte/retroarcher
```

To update the container it must be removed and recreated:

```
# Stop the container
docker stop retroarcher
# Remove the container
docker rm retroarcher
# Pull the latest update
docker pull lizardbyte/retroarcher
# Run the container with the same parameters as before
docker run -d ...
```

4.1.2 Using docker-compose

Create a `docker-compose.yml` file with the following contents (substitute your <values>):

```
version: '3'
services:
  retroarcher:
    image: lizardbyte/retroarcher
    container_name: retroarcher
    restart: unless-stopped
    volumes:
```

(continues on next page)

(continued from previous page)

```
      - <path to data>:/config
    environment:
      - PUID=<uid>
      - PGID=<gid>
      - TZ=<timezone>
    ports:
      - 9696:9696
```

Create and start the container (run the command from the same folder as your `docker-compose.yml` file):

```
docker-compose up -d
```

To update the container:

```
# Pull the latest update
docker-compose pull
# Update and restart the container
docker-compose up -d
```

4.1.3 Parameters

You must substitute the `<values>` with your own settings.

Parameters are split into two halves separated by a colon. The left side represents the host and the right side the container.

Example: `-p external:internal` - This shows the port mapping from internal to external of the container. Therefore `-p 9696:9696` would expose port `9696` from inside the container to be accessible from the host's IP on port `9696` (e.g. `http://<host_ip>:9696`). The internal port must be `9696`, but the external port may be changed (e.g. `-p 8080:9696`).

Parameter	Function	Example Value	Required
`-p <port>:9696`	Web UI Port	`9696`	True
`-v <path to data>:/config`	Volume mapping	`/home/retroarcher`	True
`-e PUID=<uid>`	User ID	`1001`	False
`-e PGID=<gid>`	Group ID	`1001`	False
`-e TZ=<timezone>`	Lookup TZ value here	`America/New_York`	True

4.1.4 User / Group Identifiers:

When using data volumes (-v flags) permissions issues can arise between the host OS and the container. To avoid this issue you can specify the user PUID and group PGID. Ensure the data volume directory on the host is owned by the same user you specify.

In this instance `PUID=1001` and `PGID=1001`. To find yours use id user as below:

```
$ id dockeruser
uid=1001(dockeruser) gid=1001(dockergroup) groups=1001(dockergroup)
```

CHAPTER

FIVE

BUILD

RetroArcher binaries are built using pyinstaller. Cross compilation is not supported. That means the binaries must be built on the target operating system and architecture.

Use Python 3.7+

5.1 Clone

Ensure git is installed and run the following:

```
git clone https://github.com/lizardbyte/retroarcher.git
cd ./retroarcher
```

5.2 Setup venv

It is recommended to setup and activate a venv within the *retroarcher* directory.

5.3 Install Python Requirements

Standard Requirements

```
python -m pip install -r requirements.txt
```

Advanced Requirements

Required for:

- *Test with flake8*
- *Test with pytest*
- *Compiling binaries*

```
python -m pip install -r requirements-dev.txt
```

Tip: Advanced requirements include all of the standard dependencies contained in the *requirements.txt*

5.4 Compile Locales

```
python ./scripts/_locale.py --compile
```

5.5 Install NPM Requirements

```
npm install
mv -f ./node_modules/ ./web/
```

5.6 Compile Docs

Docs are visible by the webapp and therefore must be compiled.

```
cd docs
make html
cd ..
```

5.7 Compile Binary

```
python ./scripts/build.py
```

5.8 Remote Build

It may be beneficial to build remotely in some cases. This will enable easier building on different operating systems.

1. Fork the project
2. Activate workflows
3. Trigger the *CI* workflow manually
4. Download the artifacts/binaries from the workflow run summary

CHAPTER

SIX

CHANGELOG

CHAPTER

SEVEN

CONTRIBUTING

Read our contribution guide in our organization level docs.

CHAPTER

EIGHT

LOCALIZATION

RetroArcher is being localized into various languages. The default language is *en* (English).

8.1 CrowdIn

The translations occur on CrowdIn. Feel free to contribute to localization there. Only elements of the interface are planned to be translated.

Translations Basics

- The brand name *LizardByte* should never be translated.
- The project name *RetroArcher* should never be translated.
- Other brand and project names should never be translated. Examples:
  - CEMU
  - GitHub
  - Linux
  - MacOS
  - RetroArch
  - RPCS3
  - Sunshine
  - Windows

CrowdIn Integration

How does it work?

When a change is made to retroarcher python source or web templates, a workflow generates new translation templates that get pushed to CrowdIn automatically.

When translations are updated on CrowdIn, a push gets made to the *l10n_master* branch and a PR is made against the *master* branch. Once PR is merged, all updated translations are part of the project and will be included in the next release.

8.2 Extraction

There should be minimal cases where strings need to be extracted from python code; however it may be necessary in some situations. For example the system tray icon is user interfacing and therefore should have strings extracted.

- **In order for strings to be extracted from python code, the following lines must be added.**

```
from pyra import locales
_ = locales.get_text()
```

- **Wrap the string to be extracted in a function as shown.**

```
_('Hello world!')
```

- **In order to include a name that should not be translated, the following example should be used.**

```
_('Open %(app_name)s') % {'app_name': 'RetroArcher'}
```

While strings are to be rarely extracted from python code, it is common for strings to be extracted from html. The extraction method from html templates is very similar to extracting from python code.

- **This is a simple *Hello world* example.**

```
{{ _('Hello world!') }}
```

- No other code needs to be added to html templates.

> **Warning:** This is for information only. Contributors should never include manually updated template files, or manually compiled language files in Pull Requests.

Strings are automatically extracted from the code to the *locale/retroarcher.po* template file. The generated file is used by CrowdIn to generate language specific template files. The file is generated using the *.github/workflows/localize.yml* workflow and is run on any push event into the *master* branch. Jobs are only run if any of the following paths are modified.

```
- 'retroarcher.py'
- 'pyra/**.py'
- 'web/templates/**'
```

When testing locally it may be desirable to manually extract, initialize, update, and compile strings.

Extract, initialize, and update

```
python ./scripts/_locale.py --extract --init --update
```

Compile

```
python ./scripts/_locale.py --compile
```

CHAPTER

NINE

TESTING

9.1 Flake8

RetroArcher uses Flake8 for enforcing consistent code styling. Flake is included in the *requirements-dev.txt*.

The config file for flake8 is *.flake8*. This is already included in the root of the repo and should not be modified.

Test with Flake8

```
python -m flake8
```

9.2 Sphinx

RetroArcher uses Sphinx for documentation building. Sphinx is included in the standard *requirements.txt* as building the docs is required for RetroArcher to be fully functional.

RetroArcher follows numpydoc styling and formatting in docstrings. This will be tested when building the docs.

The config file for Sphinx is *docs/source/conf.py*. This is already included in the root of the repo and should not be modified.

Test with Sphinx

```
cd docs
make html
```

Alternatively

```
cd docs
sphinx-build -b html source build
```

9.3 pytest

RetroArcher uses pytest for unit testing. pytest is included in the *requirements-dev.txt*.

No config is required for pytest.

Prior to running pytest you will need to:

- *compile docs*
- *compile translations*

Test with pytest

```
python -m pytest
```

CHAPTER

TEN

RETROARCHER

Responsible for starting RetroArcher.

class retroarcher.**IntRange**(*stop: int*, *start: int = 0*)

Bases: object

Custom IntRange class for argparse.

Prevents printing out large list of possible choices for integer ranges.

Parameters

stop
[int] Range maximum value.

start
[int, default = 0] Range minimum value.

Examples

```
>>> IntRange(0, 10)
<retroarcher.IntRange object at 0x...>
```

Methods

__call__:	Validate that value is within accepted range.

retroarcher.**main**()

Application entry point.

Parses arguments and initializes the application.

Examples

```
>>> if __name__ == "__main__":
...     main()
```

retroarcher.**wait**()

Wait for signal.

Endlessly loop while *pyra.SIGNAL = None*. If *pyra.SIGNAL* is changed to *shutdown* or *restart pyra.stop()* will be executed. If KeyboardInterrupt signal is detected *pyra.stop()* will be executed.

Examples

```
>>> wait()
```

CHAPTER

ELEVEN

PYRA.__INIT__

Responsible for initialization of RetroArcher.

pyra.**initialize**(*config_file: str*) → bool

Initialize RetroArcher.

Sets up config, loggers, and http port.

Parameters

config_file
[str] The path to the config file.

Returns

bool
True if initialize succeeds, otherwise False.

Raises

SystemExit
If unable to correct possible issues with config file.

Examples

```
>>> initialize(config_file='config.ini')
True
```

pyra.**stop**(*exit_code: int | str = 0*, *restart: bool = False*)

Stop RetroArcher.

This function ends the tray icon if it's running. Then restarts or shutdowns RetroArcher depending on the value of the *restart* parameter.

Parameters

exit_code
[Union[int, str], default = 0] The exit code to send. Does not apply if *restart = True*.

restart
[bool, default = False] Set to True to restart RetroArcher.

Examples

```
>>> stop(exit_code=0, restart=False)
```

CHAPTER

TWELVE

PYRA.CONFIG

Responsible for config related functions.

pyra.config.**convert_config**(*d: dict = _CONFIG_SPEC_DICT*, *_config_spec: List | None = None*) → List

Convert a config spec dictionary to a config spec list.

A config spec dictionary is a custom type of dictionary that will be converted into a standard config spec list which can later be used by `configobj`.

Parameters

d
[dict] The dictionary to convert.

_config_spec
[Optional[List]] This should not be set when using this function, but since this function calls itself it needs to pass in the list that is being built in order to return the correct list.

Returns

list
A list representing a configspec for `configobj`.

Examples

```
>>> convert_config(d=_CONFIG_SPEC_DICT)
[...]
```

pyra.config.**create_config**(*config_file: str*, *config_spec: dict = _CONFIG_SPEC_DICT*) → ConfigObj

Create a config file and *ConfigObj* using a config spec dictionary.

A config spec dictionary is a strictly formatted dictionary that will be converted into a standard config spec list to be later used by `configobj`.

The created config is validated against a Validator object. This function will remove keys from the user's config.ini if they no longer exist in the config spec.

Parameters

config_file
[str] Full filename of config file.

config_spec
[dict, default = _CONFIG_SPEC_DICT] Config spec to use.

Returns

ConfigObj
Dictionary of config keys and values.

Raises

SystemExit
If config_spec is not valid.

Examples

```
>>> create_config(config_file='config.ini')
ConfigObj({...})
```

pyra.config.**on_change_tray_toggle**() → bool

Toggle the tray icon.

This is needed, since `tray_icon` cannot be imported at the module level without a circular import.

Returns

bool
`True` if successful, otherwise `False`.

See also:

pyra.tray_icon.tray_toggle
`on_change_tray_toggle` is an alias of this function.

Examples

```
>>> on_change_tray_toggle()
True
```

pyra.config.**save_config**(*config: ConfigObj | None = CONFIG*) → bool

Save the config to file.

Saves the *ConfigObj* to the specified file.

Parameters

config
[ConfigObj, default = CONFIG] Config to save.

Returns

bool
True if save successful, otherwise False.

Examples

```
>>> config_object = create_config(config_file='config.ini')
>>> save_config(config=config_object)
True
```

pyra.config.**validate_config**(*config: ConfigObj*) → bool

Validate ConfigObj dictionary.

Ensures that the given *ConfigObj* is valid.

Parameters

config
[ConfigObj] Config to validate.

Returns

bool
True if validation passes, otherwise False.

Examples

```
>>> config_object = create_config(config_file='config.ini')
>>> validate_config(config=config_object)
True
```

CHAPTER

THIRTEEN

PYRA.DEFINITIONS

Contains classes with attributes to common definitions (paths and filenames).

class pyra.definitions.**Files**

Bases: `object`

Class representing common Files.

The purpose of this class is to ensure consistency when using these files.

CONFIG
[str] The default config file name. i.e. *config.ini*.

Examples

```
>>> Files.CONFIG
'config.ini'
```

class pyra.definitions.**Modes**

Bases: `object`

Class representing runtime variables.

FROZEN
[bool] `True` if running pyinstaller bundle version, otherwise `False`.

DOCKER
[bool] `True` if running Docker version, otherwise `False`.

SPLASH
[bool] `True` if capable of displaying a splash image on start, otherwise, `False`.

Examples

```
>>> Modes.FROZEN
False
```

class pyra.definitions.**Names**

Bases: `object`

Class representing common names.

The purpose of this class is to ensure consistency when using these names.

name
[str] The application's name. i.e. *RetroArcher*.

Examples

```
>>> Names.name
'RetroArcher'
```

class pyra.definitions.**Paths**

Bases: object

Class representing common Paths.

The purpose of this class is to ensure consistency when using these paths.

PYRA_DIR
[str] The directory containing the retroarcher python files.

ROOT_DIR
[str] The root directory of the application. This is where the source files exist.

DATA_DIR
[str] The data directory of the application.

DOCS_DIR
[str] The directory containing html documentation.

LOCALE_DIR
[str] The directory containing localization files.

LOG_DIR
[str] The directory containing log files.

Examples

```
>>> Paths.logs
'.../logs'
```

class pyra.definitions.**Platform**

Bases: object

Class representing the machine platform.

The purpose of this class is to ensure consistency when there is a need for platform specific functions.

bits
[str] Operating system bitness. e.g. 64.

operating_system
[str] Operating system name. e.g. 'Windows'.

os_platform
[str] Operating system platform. e.g. 'win32', 'darwin', 'linux'.

machine
[str] Machine architecture. e.g. 'AMD64'.

node
[str] Machine name.

release
[str] Operating system release. e.g. '10'.

version
[str] Operating system version. e.g. '10.0.22000'.

edition
[str] Windows edition. e.g. 'Core', None for non Windows platforms.

iot
[bool] True if Windows IOT, otherwise False.

Examples

```
>>> Platform.os_platform
...
```

Attributes

edition

CHAPTER

FOURTEEN

PYRA.HARDWARE

Functions related to the dashboard viewer.

pyra.hardware.**chart_data**() → dict

Get chart data.

Get the data from the `dash_stats` dictionary, formatted for use with `plotly`.

Returns

dict
A single key named 'graphs' contains a list of graphs. Each graph is formatted as a dictionary and ready to use with `plotly`.

See also:

pyra.webapp.callback_dashboard
A callback called by javascript to get this data.

Examples

```
>>> chart_data()
{'graphs': [{"data": [...], "layout": ..., "config": ..., {"data": ...]}
```

pyra.hardware.**chart_types**()

Get chart types.

Get the type of charts supported by the system.

Returns

list
A list containing the types of charts supported.

Examples

```
>>> chart_types()
['cpu', 'memory', 'network']
```

```
>>> chart_types()
['cpu', 'gpu', 'memory', 'network']
```

pyra.hardware.**update**()

Update all dashboard stats.

This function updates the cpu and memory usage of this python process as well as subprocesses. Following that the system functions are called to update system cpu, gpu, memory, and network usage. Finally, the keys in the `dash_stats` dictionary are cleaned up to only hold 120 values. This function is called once per second, therefore there are 2 minutes worth of values in the dictionary.

Examples

```
>>> update()
```

pyra.hardware.**update_cpu**() → float

Update dashboard stats for system CPU usage.

This will append a new value to the `dash_stats['cpu'][system']` list.

Returns

float
The current system cpu percentage utilized.

Examples

```
>>> update_cpu()
```

pyra.hardware.**update_gpu**()

Update dashboard stats for system GPU usage.

This will create new keys for the `dash_stats` dictionary if required, and then append a new value to the appropriate list.

AMD data is provided by pyamdgpuinfo on Linux, and by pyadl on non Linux systems. Nvidia data is provided by GPUtil.

Examples

```
>>> update_gpu()
```

pyra.hardware.**update_memory**()

Update dashboard stats for system memory usage.

This will append a new value to the `dash_stats['memory']['system']` list.

Returns

float
The current system memory percentage utilized.

Examples

```
>>> update_memory()
```

pyra.hardware.**update_network**()

Update dashboard stats for system network usage.

This will append a new values to the `dash_stats['network']['received']` and `dash_stats['network']['sent']` lists.

Returns

tuple
A tuple of the received and sent values as a difference since the last update.

Examples

```
>>> update_network()
```

CHAPTER

FIFTEEN

PYRA.HELPERS

Many reusable helper functions.

pyra.helpers.**check_folder_writable**(*fallback: str*, *name: str*, *folder: str | None = None*) → tuple[str, bool | None]

Check if folder or fallback folder is writeable.

This function ensures that the folder can be created, if it doesn't exist. It also ensures there are sufficient permissions to write to the folder. If the primary *folder* fails, it falls back to the *fallback* folder.

Parameters

fallback
[str] Secondary folder to check, if the primary folder fails.

name
[str] Short name of folder.

folder
[str, optional] Primary folder to check.

Returns

tuple[str, Optional[bool]]

A tuple containing:

folder
[str] The original or fallback folder.

Optional[bool]
True if writeable, otherwise False. Nothing is returned if there is an error attempting to create the directory.

Examples

```
>>> check_folder_writable(
...     folder='logs',
...     fallback='backup_logs',
...     name='logs'
...     )
('logs', True)
```

pyra.helpers.**docker_healthcheck**() → bool

Check the health of the docker container.

> **Warning:** This is only meant to be called by *retroarcher.py*, and the interpreter should be immediate exited following the result.

The default port is used considering that the container will use the default port internally. The external port should not make any difference.

Returns

bool
True if status okay, otherwise False.

Examples

```
>>> docker_healthcheck()
True
```

pyra.helpers.**get_ip**(*host: str*) → str | None

Get IP address from host name.

This function is used to get the IP address of a given host name.

Parameters

host
[str] Host name to get ip address of.

Returns

str
IP address of host name if it is a valid ip address, otherwise `None`.

Examples

```
>>> get_ip(host='192.168.1.1')
'192.168.1.1'
```

```
>>> get_ip(host='www.google.com')
'172.253.63.147'
```

pyra.helpers.**get_logger**(*name: str*) → Logger

Get the logger for the given name.

This function also exists in *logger.py* to prevent circular imports.

Parameters

name
[str] Name of logger.

Returns

logging.Logger
The logging.Logger object.

Examples

```
>>> get_logger(name='my_log')
<Logger my_log (WARNING)>
```

pyra.helpers.**is_public_ip**(*host: str*) → bool

Check if ip address is public or not.

This function is used to determine if the given host address is a public ip address or not.

Parameters

host
[str] IP address to check.

Returns

bool
True if ip address is public, otherwise False.

Examples

```
>>> is_public_ip(host='www.google.com')
True
```

```
>>> is_public_ip(host='192.168.1.1')
False
```

pyra.helpers.**is_valid_ip**(*address: str*) → IP | bool

Check if address is an ip address.

This function is used to determine if the given address is an ip address or not.

Parameters

address
[str] Address to check.

Returns

Union[IP, bool]
IP object if address is an ip address, otherwise False.

Examples

```
>>> is_valid_ip(address='192.168.1.1')
True
```

```
>>> is_valid_ip(address='0.0.0.0.0')
False
```

pyra.helpers.**now**(*separate: bool = False*) → str

Function to get the current time, formatted.

This function will return the current time formatted as YMDHMS

Parameters

separate
[bool, default = False] True to separate time with a combination of dashes (-) and colons (:).

Returns

str
The current time formatted as YMDHMS.

Examples

```
>>> now()
'20220410184531'
```

```
>>> now(separate=True)
'2022-04-10 18:46:12'
```

pyra.helpers.**open_url_in_browser**(*url: str*) → bool

Open a given url in the default browser.

Attempt to open the given url in the default web browser, in a new tab.

Parameters

url
[str] The url to open.

Returns

bool
True if no error, otherwise False.

Examples

```
>>> open_url_in_browser(url='https://www.google.com')
True
```

pyra.helpers.**timestamp**() → int

Function to get the current time.

This function uses time.time() to get the current time.

Returns

int
The current time as a timestamp integer.

Examples

```
>>> timestamp()
1649631005
```

pyra.helpers.**timestamp_to_YMDHMS**(*ts: int*, *separate: bool = False*) → str

Convert timestamp to YMDHMS format.

Convert a given timestamp to YMDHMS format.

Parameters

ts
[int] The timestamp to convert.

separate
[bool, default = False] True to separate time with a combination of dashes (-) and colons (:).

Returns

str
The timestamp formatted as YMDHMS.

Examples

```
>>> timestamp_to_YMDHMS(ts=timestamp(), separate=False)
'20220410185142'
```

```
>>> timestamp_to_YMDHMS(ts=timestamp(), separate=True)
'2022-04-10 18:52:09'
```

pyra.helpers.**timestamp_to_datetime**(*ts: float*) → datetime

Convert timestamp to datetime object.

This function returns the result of *datetime.datetime.fromtimestamp()*.

Parameters

ts
[float] The timestamp to convert.

Returns

datetime.datetime
Object *datetime.datetime*.

Examples

```
>>> timestamp_to_datetime(ts=timestamp())
datetime.datetime(20..., ..., ..., ..., ..., ...)
```

CHAPTER

SIXTEEN

PYRA.LOCALES

Functions related to localization.

Localization (also referred to as l10n) is the process of adapting a product or service to a specific locale. Translation is only one of several elements in the localization process. In addition to translation, the localization process may also include: - Adapting design and layout to properly display translated text in the language of the locale - Adapting sorting functions to the alphabetical order of a specific locale - Changing formats for date and time, addresses, numbers, currencies, etc. for specific target locales - Adapting graphics to suit the expectations and tastes of a target locale - Modifying content to suit the tastes and consumption habits of a target locale

The aim of localization is to give a product or service the look and feel of having been created specifically for a target market, no matter their language, cultural preferences, or location.

pyra.locales.**get_all_locales**() → dict

Get a dictionary of all possible locales for use with babel.

Dictionary keys will be *locale_id* and value with be *locale_display_name*. This is a shortened example of the returned value.

```
{
  'de': 'Deutsch',
  'en': 'English',
  'en_GB': 'English (United Kingdom)',
  'en_US': 'English (United States)',
  'es': 'español',
  'fr': 'français',
  'it': 'italiano',
  'ru': ''
}
```

Returns

dict
Dictionary of all possible locales.

Examples

```
>>> get_all_locales()
{... 'en': 'English', ... 'en_GB': 'English (United Kingdom)', ... 'es': 'español',␣
↪... 'fr': 'français', ...}
```

pyra.locales.**get_locale**() → str

Verify the locale.

Verify the locale from the config against supported locales and returns appropriate locale.

Returns

str
The locale set in the config if it is valid, otherwise the default locale (en).

Examples

```
>>> get_locale()
'en'
```

pyra.locales.**get_text**() → gettext

Install the language defined in the conifg.

This function installs the language defined in the config and allows translations in python code.

Returns

gettext.gettext
The *gettext.gettext* method.

Examples

```
>>> get_text()
<bound method GNUTranslations.gettext of <gettext.GNUTranslations object at 0x...>>
```

CHAPTER

SEVENTEEN

PYRA.LOGGER

Responsible for logging related functions.

class pyra.logger.**BlacklistFilter**

Bases: `Filter`

Filter logs for blacklisted words.

Log filter for blacklisted tokens and passwords.

Examples

```
>>> BlacklistFilter()
<pyra.logger.BlacklistFilter object at 0x...>
```

Methods

filter:	Filter the given record.

filter(*record*) → bool

Filter the given record.

Todo: This documentation needs to be improved.

Parameters

record
[BlacklistFilter] The record to filter.

Returns

bool
True in all cases.

Examples

```
>>> BlacklistFilter().filter(record=BlacklistFilter())
True
```

class pyra.logger.**EmailFilter**

Bases: *RegexFilter*

Log filter for email addresses.

Class responsible for filtering email addresses.

Examples

```
>>> EmailFilter()
<pyra.logger.EmailFilter object at 0x...>
```

Attributes

regex
[re.compile] The compiled regex pattern.

Methods

replace:	Filter that replaces a string within another string.

replace(*text: str*, *email: str*) → str

Filter an email address.

Filter the given email address out of the given text.

Parameters

text
[str] The text to replace the email address within.

email
[str] The email address to replace with asterisks.

Returns

str
The original text with the email address replaced.

Examples

```
>>> EmailFilter().replace(text='Testing example@example.com', email=
↪'example@example.com')
'Testing ****************@********'
```

class pyra.logger.**NoThreadFilter**(*threadName*)

Bases: `Filter`

Log filter for the current thread.

Todo: This documentation needs to be improved.

Parameters

threadName
[str] The name of the thread.

Examples

```
>>> NoThreadFilter('main')
<pyra.logger.NoThreadFilter object at 0x...>
```

Methods

filter:	Filter the given record.

filter(*record*) → bool

Filter the given record.

Todo: This documentation needs to be improved.

Parameters

record
[NoThreadFilter] The record to filter.

Returns

bool
True if record.threadName is not equal to self.threadName, otherwise False.

Examples

```
>>> NoThreadFilter('main').filter(record=NoThreadFilter('test'))
True
```

```
>>> NoThreadFilter('main').filter(record=NoThreadFilter('main'))
False
```

class pyra.logger.**PlexTokenFilter**

Bases: *RegexFilter*

Log filter for X-Plex-Token.

Class responsible for filtering Plex tokens.

Examples

```
>>> PlexTokenFilter()
<pyra.logger.PlexTokenFilter object at 0x...>
```

Attributes

regex
[re.compile] The compiled regex pattern.

Methods

replace:	Filter that replaces a string within another string.

replace(*text: str*, *token: str*) → str

Filter a token.

Filter the given token out of the given text.

Parameters

text
[str] The text to replace the token within.

token
[str] The token to replace with asterisks.

Returns

str
The original text with the token replaced.

Examples

```
>>> PlexTokenFilter().replace(text='x-plex-token=5FBCvHo9vFf9erz8ssLQ', token=
↪'5FBCvHo9vFf9erz8ssLQ')
'x-plex-token=**********'
```

class pyra.logger.**PublicIPFilter**

Bases: *RegexFilter*

Log filter for public IP addresses.

Class responsible for filtering public IP addresses.

Examples

```
>>> PublicIPFilter()
<pyra.logger.PublicIPFilter object at 0x...>
```

Attributes

regex
[re.compile] The compiled regex pattern.

Methods

replace:	Filter that replaces a string within another string.

replace(*text: str*, *ip: str*) → str

Filter a public address.

Filter the given ip address out of the given text. The ip address will only be filter if it is public.

Parameters

text
[str] The text to replace the ip address within.

ip
[str] The ip address to replace with asterisks.

Returns

str
The original text with the ip address replaced.

Examples

```
>>> PublicIPFilter().replace(text='Testing 172.1.7.5', ip='172.1.7.5')
'Testing ***.***.***.***'
```

class pyra.logger.**RegexFilter**

Bases: `Filter`

Base class for regex log filter.

Log filter for regex.

Examples

```
>>> RegexFilter()
<pyra.logger.RegexFilter object at 0x...>
```

Attributes

regex
[re.compile] The compiled regex pattern.

Methods

filter:	Filter the given record.

filter(*record*) → bool

Filter the given record.

Todo: This documentation needs to be improved.

Parameters

record
[RegexFilter] The record to filter.

Returns

bool
True in all cases.

Examples

```
>>> RegexFilter().filter(record=RegexFilter())
True
```

pyra.logger.**blacklist_config**(*config: ConfigObj*)

Update blacklist words.

In order to filter words out of the logs, it is required to call this function.

Values in the config for keys containing the following terms will be removed.

- HOOK
- APIKEY
- KEY
- PASSWORD
- TOKEN

Parameters

config
[ConfigObj] Config to parse.

Examples

```
>>> config_object = pyra.config.create_config(config_file='config.ini')
>>> blacklist_config(config=config_object)
```

pyra.logger.**get_logger**(*name: str*) → Logger

Get a logger.

Return the logging.Logger object for a given name. Additionally, replaces logger.warn with logger.warning.

Parameters

name
[str] The name of the logger to get.

Returns

logging.Logger
The logging.Logger object.

Examples

```
>>> get_logger(name='retroarcher')
<Logger retroarcher (WARNING)>
```

pyra.logger.**init_logger**(*log_name: str*) → Logger

Create a logger.

Creates a logging.Logger object from the given log name.

Parameters

log_name
[str] The name of the log to create.

Returns

logging.Logger
The logging.Logger object.

Examples

```
>>> init_logger(log_name='retroarcher')
<Logger retroarcher (INFO)>
```

pyra.logger.**init_multiprocessing**(*logger: Logger*)

Remove all handlers and add QueueHandler on top.

This should only be called inside a multiprocessing worker process, since it changes the logger completely.

Parameters

logger
[logging.Logger] The logger to initialize for multiprocessing.

Examples

```
>>> logger = get_logger(name='retroarcher')
>>> init_multiprocessing(logger=logger)
```

pyra.logger.**listener**(*logger: Logger*)

Create a QueueListener.

Wrapper that create a QueueListener, starts it and automatically stops it. To be used in a with statement in the main process, for multiprocessing.

Parameters

logger
[logging.Logger] The logger object.

Examples

```
>>> logger = get_logger(name='retroarcher')
>>> listener(logger=logger)
```

pyra.logger.**setup_loggers**()

Setup all loggers.

Setup all the available loggers.

Examples

```
>>> setup_loggers()
```

pyra.logger.**shutdown**()

Stop logging.

Shutdown logging.

Examples

```
>>> shutdown()
```

CHAPTER

EIGHTEEN

PYRA.THREADS

Functions related to threading.

18.1 Routine Listings

run_in_thread
[method] Alias of the built in method *threading.Thread*.

18.2 Examples

```
>>> from pyra import config, threads, tray_icon
>>> config_object = config.create_config(config_file='config.ini')
>>> tray_icon.icon = tray_icon.tray_initialize()
>>> threads.run_in_thread(target=tray_icon.tray_run, name='pystray', daemon=True).start()
```

```
>>> from pyra import config, threads, webapp
>>> config_object = config.create_config(config_file='config.ini')
>>> threads.run_in_thread(target=webapp.start_webapp, name='Flask', daemon=True).start()
 * Serving Flask app 'pyra.webapp' (lazy loading)
...
 * Running on http://.../ (Press CTRL+C to quit)
```

CHAPTER

NINETEEN

PYRA.TRAY_ICON

Responsible for system tray icon and related functions.

pyra.tray_icon.**donate_github**()

Open GitHub Sponsors.

Open GitHub Sponsors in the default web browser.

Returns

bool
True if opening page was successful, otherwise False.

Examples

```
>>> donate_github()
True
```

pyra.tray_icon.**donate_mee6**()

Open MEE6.

Open MEE6 in the default web browser.

Returns

bool
True if opening page was successful, otherwise False.

Examples

```
>>> donate_mee6()
True
```

pyra.tray_icon.**donate_patreon**()

Open Patreon.

Open Patreon in the default web browser.

Returns

bool
True if opening page was successful, otherwise False.

Examples

```
>>> donate_patreon()
True
```

pyra.tray_icon.**donate_paypal**()

Open PayPal.

Open PayPal in the default web browser.

Returns

bool
True if opening page was successful, otherwise False.

Examples

```
>>> donate_paypal()
True
```

pyra.tray_icon.**github_releases**()

Open GitHub Releases.

Open GitHub Releases in the default web browser.

Returns

bool
True if opening page was successful, otherwise False.

Examples

```
>>> github_releases()
True
```

pyra.tray_icon.**open_webapp**() → bool

Open the webapp.

Open RetroArcher in the default web browser.

Returns

bool
True if opening page was successful, otherwise False.

Examples

```
>>> open_webapp()
True
```

pyra.tray_icon.**tray_browser**()

Toggle the config option 'LAUNCH_BROWSER'.

This functions switches the *LAUNCH_BROWSER* config option from True to False, or False to True.

Examples

```
>>> tray_browser()
```

pyra.tray_icon.**tray_disable**()

Turn off the config option 'SYSTEM_TRAY'.

This function ends and disables the *SYSTEM_TRAY* config option.

Examples

```
>>> tray_disable()
```

pyra.tray_icon.**tray_end**() → bool

End the system tray icon.

Hide and then stop the system tray icon.

Returns

bool
`True` if successful, otherwise `False`.

Examples

```
>>> tray_end()
```

pyra.tray_icon.**tray_initialize**() → None | bool

Initialize the system tray icon.

Some features of the tray icon may not be available, depending on the operating system. An attempt is made to setup the tray icon with all the available features supported by the OS.

Returns

Union[Icon, bool]

Icon
Instance of pystray.Icon if icon is supported.

bool
False if icon is not supported.

Examples

```
>>> tray_initialize()
```

pyra.tray_icon.**tray_quit**()

Shutdown RetroArcher.

Set the 'pyra.SIGNAL' variable to 'shutdown'.

Examples

```
>>> tray_quit()
```

pyra.tray_icon.**tray_restart**()

Restart RetroArcher.

Set the 'pyra.SIGNAL' variable to 'restart'.

Examples

```
>>> tray_restart()
```

pyra.tray_icon.**tray_run**()

Start the tray icon.

Run the system tray icon in detached mode.

Examples

```
>>> tray_run()
```

pyra.tray_icon.**tray_run_threaded**() → bool

Run the system tray in a thread.

This function exectues various other functions to simplify starting the tray icon.

Returns

bool
`True` if successful, otherwise `False`.

See also:

tray_initialize
This function first, initializes the tray icon using `tray_initialize()`.

tray_run
Then, `tray_run` is executed in a thread.

pyra.threads.run_in_thread
Run a method within a thread.

Examples

```
>>> tray_run_threaded()
True
```

pyra.tray_icon.**tray_toggle**() → bool

Toggle the system tray icon.

Hide/unhide the system tray icon.

Returns

bool
`True` if successful, otherwise `False`.

Examples

```
>>> tray_toggle()
```

CHAPTER

TWENTY

PYRA.WEBAPP

Responsible for serving the webapp.

pyra.webapp.**api_settings**(*configuration_spec: str | None*) → Response

Get current settings or save changes to settings from web ui.

This endpoint accepts a *GET* or *POST* request. A *GET* request will return the current settings. A *POST* request will process the data passed in and return the results of processing.

Parameters

configuration_spec
[Optional[str]] The spec to return. In the future this will be used to return config specs of plugins; however that is not currently implemented.

Returns

Response
A response formatted as `flask.jsonify`.

Examples

```
>>> callback_dashboard()
<Response ... bytes [200 OK]>
```

pyra.webapp.**callback_dashboard**() → Response

Get dashboard data.

This should be used in a callback in order to update charts in the web app.

Returns

Response
A response formatted as `flask.jsonify`.

See also:

pyra.hardware.chart_data
This function sets up the data in the proper format.

Examples

```
>>> callback_dashboard()
<Response ... bytes [200 OK]>
```

pyra.webapp.**docs**(*filename*) → send_from_directory

Serve the Sphinx html documentation.

Todo: This documentation needs to be improved.

Parameters

filename
[str] The html filename to return.

Returns

flask.send_from_directory
The requested documentation page.

Notes

The following routes trigger this function.

/docs/ /docs/<page.html>

Examples

```
>>> docs(filename='index.html')
```

pyra.webapp.**favicon**() → send_from_directory

Serve the favicon.ico file.

Todo: This documentation needs to be improved.

Returns

flask.send_from_directory
The ico file.

Notes

The following routes trigger this function.

/favicon.ico

Examples

```
>>> favicon()
```

pyra.webapp.**home**() → render_template

Serve the webapp home page.

Todo: This documentation needs to be improved.

Returns

render_template
The rendered page.

Notes

The following routes trigger this function.

/ /home

Examples

```
>>> home()
```

pyra.webapp.**render_template**(*template_name_or_list*, ***context*)

Render a template, while providing our default context.

This function is a wrapper around `flask.render_template`. Our UI config is added to the template context. In the future, this function may be used to add other default contexts to templates.

Parameters

template_name_or_list
[str] The name of the template to render.

****context**
The context to pass to the template.

Returns

render_template
The rendered template.

Examples

```
>>> render_template(template_name_or_list='home.html', title=_('Home'))
```

pyra.webapp.**settings**(*configuration_spec: str | None*) → render_template

Serve the configuration page page.

Todo: This documentation needs to be improved.

Parameters

configuration_spec
[Optional[str]] The spec to return. In the future this will be used to return config specs of plugins; however that is not currently implemented.

Returns

render_template
The rendered page.

Notes

The following routes trigger this function.

/settings

Examples

```
>>> settings()
```

pyra.webapp.**start_webapp**()

Start the webapp.

Start the flask webapp. This is placed in it's own function to allow the ability to start the webapp within a thread in a simple way.

Examples

```
>>> start_webapp()
 * Serving Flask app 'pyra.webapp' (lazy loading)
...
 * Running on http://.../ (Press CTRL+C to quit)
```

```
>>> from pyra import webapp, threads
>>> threads.run_in_thread(target=webapp.start_webapp, name='Flask', daemon=True).
↪start()
 * Serving Flask app 'pyra.webapp' (lazy loading)
...
 * Running on http://.../ (Press CTRL+C to quit)
```

pyra.webapp.**status**() → dict

Check the status of RetroArcher.

This is useful for a healthcheck from Docker, and may have many other uses in the future for third party applications.

Returns

dict
A dictionary of the status.

Examples

```
>>> status()
```

pyra.webapp.**test_logger**() → str

Test logging functions.

Check *./logs/pyra.webapp.log* for output.

Returns

str
A message telling the user to check the logs.

Notes

The following routes trigger this function.

/test_logger

Examples

```
>>> test_logger()
```

PYTHON MODULE INDEX

p

pyra, 23
pyra.config, 25
pyra.definitions, 29
pyra.hardware, 33
pyra.helpers, 37
pyra.locales, 43
pyra.logger, 45
pyra.threads, 55
pyra.tray_icon, 57
pyra.webapp, 63

r

retroarcher, 21

INDEX

A

api_settings() (*in module pyra.webapp*), 63

B

blacklist_config() (*in module pyra.logger*), 51
BlacklistFilter (*class in pyra.logger*), 45

C

callback_dashboard() (*in module pyra.webapp*), 63
chart_data() (*in module pyra.hardware*), 33
chart_types() (*in module pyra.hardware*), 33
check_folder_writable() (*in module pyra.helpers*), 37
convert_config() (*in module pyra.config*), 25
create_config() (*in module pyra.config*), 25

D

docker_healthcheck() (*in module pyra.helpers*), 37
docs() (*in module pyra.webapp*), 64
donate_github() (*in module pyra.tray_icon*), 57
donate_mee6() (*in module pyra.tray_icon*), 57
donate_patreon() (*in module pyra.tray_icon*), 57
donate_paypal() (*in module pyra.tray_icon*), 58

E

EmailFilter (*class in pyra.logger*), 46

F

favicon() (*in module pyra.webapp*), 64
Files (*class in pyra.definitions*), 29
filter() (*pyra.logger.BlacklistFilter method*), 45
filter() (*pyra.logger.NoThreadFilter method*), 47
filter() (*pyra.logger.RegexFilter method*), 50

G

get_all_locales() (*in module pyra.locales*), 43
get_ip() (*in module pyra.helpers*), 38
get_locale() (*in module pyra.locales*), 44
get_logger() (*in module pyra.helpers*), 38
get_logger() (*in module pyra.logger*), 51
get_text() (*in module pyra.locales*), 44
github_releases() (*in module pyra.tray_icon*), 58

H

home() (*in module pyra.webapp*), 65

I

init_logger() (*in module pyra.logger*), 51
init_multiprocessing() (*in module pyra.logger*), 52
initialize() (*in module pyra*), 23
IntRange (*class in retroarcher*), 21
is_public_ip() (*in module pyra.helpers*), 39
is_valid_ip() (*in module pyra.helpers*), 39

L

listener() (*in module pyra.logger*), 52

M

main() (*in module retroarcher*), 21
Modes (*class in pyra.definitions*), 29
module
    pyra, 23
    pyra.config, 25
    pyra.definitions, 29
    pyra.hardware, 33
    pyra.helpers, 37
    pyra.locales, 43
    pyra.logger, 45
    pyra.threads, 55
    pyra.tray_icon, 57
    pyra.webapp, 63
    retroarcher, 21

N

Names (*class in pyra.definitions*), 29
NoThreadFilter (*class in pyra.logger*), 47
now() (*in module pyra.helpers*), 39

O

on_change_tray_toggle() (*in module pyra.config*), 26
open_url_in_browser() (*in module pyra.helpers*), 40
open_webapp() (*in module pyra.tray_icon*), 58

P

Paths (*class in pyra.definitions*), 30
Platform (*class in pyra.definitions*), 30
PlexTokenFilter (*class in pyra.logger*), 48
PublicIPFilter (*class in pyra.logger*), 49
pyra
    module, 23
pyra.config
    module, 25
pyra.definitions
    module, 29
pyra.hardware
    module, 33
pyra.helpers
    module, 37
pyra.locales
    module, 43
pyra.logger
    module, 45
pyra.threads
    module, 55
pyra.tray_icon
    module, 57
pyra.webapp
    module, 63

R

RegexFilter (*class in pyra.logger*), 50
render_template() (*in module pyra.webapp*), 65
replace() (*pyra.logger.EmailFilter method*), 46
replace() (*pyra.logger.PlexTokenFilter method*), 48
replace() (*pyra.logger.PublicIPFilter method*), 49
retroarcher
    module, 21

S

save_config() (*in module pyra.config*), 26
settings() (*in module pyra.webapp*), 65
setup_loggers() (*in module pyra.logger*), 52
shutdown() (*in module pyra.logger*), 53
start_webapp() (*in module pyra.webapp*), 66
status() (*in module pyra.webapp*), 66
stop() (*in module pyra*), 23

T

test_logger() (*in module pyra.webapp*), 67
timestamp() (*in module pyra.helpers*), 40
timestamp_to_datetime() (*in module pyra.helpers*), 41
timestamp_to_YMDHMS() (*in module pyra.helpers*), 41
tray_browser() (*in module pyra.tray_icon*), 58
tray_disable() (*in module pyra.tray_icon*), 59
tray_end() (*in module pyra.tray_icon*), 59
tray_initialize() (*in module pyra.tray_icon*), 59
tray_quit() (*in module pyra.tray_icon*), 59
tray_restart() (*in module pyra.tray_icon*), 60
tray_run() (*in module pyra.tray_icon*), 60
tray_run_threaded() (*in module pyra.tray_icon*), 60
tray_toggle() (*in module pyra.tray_icon*), 60

U

update() (*in module pyra.hardware*), 33
update_cpu() (*in module pyra.hardware*), 34
update_gpu() (*in module pyra.hardware*), 34
update_memory() (*in module pyra.hardware*), 34
update_network() (*in module pyra.hardware*), 35

V

validate_config() (*in module pyra.config*), 27

W

wait() (*in module retroarcher*), 22