# Universal Downloader

***Release 2.0.0***

**Iceflower S**

**Sep 08, 2020**

# CONTENTS

CHAPTER

ONE

# USER GUIDE

## 1.1 Installation

### 1.1.1 System Requirements

Python is required with version 3.8. or higher, it can be downloaded at https://www.python.org/downloads/.

During the Windows installation you should make sure that the `PATH` / environment variable is set and `pip` is installed.

Under Linux it should be ensured that `pip` is installed, if this is not done within the standard installation.

### 1.1.2 Installation

The installation is preferably done over the terminal.

#### Pip

Installation with pip.

```
pip install unidown
```

## 1.2 Using UniDown

The program is a terminal program, so it runs from the terminal.

Calling with:

```
unidown
```

Furthermore, there are additional arguments:

**-h, --help**
show this help message and exit

**-v, --version**
show program's version number and exit

**--list-plugins**
show plugin list and exit

**-p** name, **--plugin** name
plugin to execute

**-o** option [option ...], **--option** option [option ...]
options passed to the plugin, e.g. *-o username=South American coati -o password=Nasua Nasua*

**--logfile** path
log filepath relativ to the main dir (default: ./unidown.log)

**-l** {DEBUG,INFO,WARNING,ERROR,CRITICAL}, **--log** {DEBUG,INFO,WARNING,ERROR,CRITICAL}
set the logging level (default: INFO)

CHAPTER

TWO

# WRITING PLUGINS

## 2.1 Structure

### 2.1.1 Variables

**_info** information about the plugin, must be set everytime

**_savestate_cls** must be set if a custom SaveState format is in use

**_simul_downloads** adjust it to a low value to reduce the load on the target server

**_options** options, passed by the command line, `delay` will be set to 0 in absence of a value, set it to a higher value to reduce the load on the target server

**_unit** unit displayed while downloading

### 2.1.2 Methods

**_create_last_update_time** returns the update time of the complete data e.g. the newest item in the collection If for some reasons its not easily collectable or not available or want to check the links every time, return the current time.

**_create_download_data** returns a LinkItemDict, with links and their update time

**_load_default_options** override if you need your own default options

**load_savestate** override if you have your own custom savestate

**update_savestate** override if you have your own custom savestate

### 2.1.3 LinkItemDict

The LinkItemDict is an essential part of unidown. It is a normal dictionary with some special function. The key is the link as a string. The value is a LinkItem.

### 2.1.4 LinkItem

LinkItem has two essential values `name` and `time`, the used name is at the same time the file given at downloading.

## 2.2 Hello World

We will use the test plugin, to show how to create a plugin for unidown.

### 2.2.1 Basics

First of all we subclass from *unidown.plugin.a_plugin.APlugin*, to have all essential variables and functions for free.

```
from unidown.plugin import APlugin

class Plugin(APlugin):
```

Next part is to add basic information about the plugin like name, version and server host.

```
class Plugin(APlugin):
    _info = PluginInfo('test', '0.1.0', 'raw.githubusercontent.com')
```

Next we will hook into the options loading to set default options if nothing was passed.

```
class Plugin(APlugin):
    _info = PluginInfo('test', '0.1.0', 'raw.githubusercontent.com')

def _load_default_options(self):
    super(Plugin, self)._load_default_options()
    if 'username' not in self._options:
        self._options['username'] = ''
```

Now as the username is in the options for sure we can just get it safely.

```
class Plugin(APlugin):
    _info = PluginInfo('test', '0.1.0', 'raw.githubusercontent.com')

def __init__(self, settings: Settings, options: Dict[str, Any] = None):
    super().__init__(settings, options)
    self._username: str = self._options['username']

def _load_default_options(self):
    super(Plugin, self)._load_default_options()
    if 'username' not in self._options:
        self._options['username'] = ''
```

To get an overall update time for the complete data set, create the `_create_last_update_time` method. If the time generated by this function is older compared to the savestate it will run the individual link generating, if not it will skip the rest.

```
class Plugin(APlugin):
    _info = PluginInfo('test', '0.1.0', 'raw.githubusercontent.com')

def __init__(self, settings: Settings, options: Dict[str, Any] = None):
```

(continues on next page)

(continued from previous page)

```
    super().__init__(settings, options)
    self._username: str = self._options['username']

def _create_last_update_time(self) -> datetime:
    self.download_as_file('/IceflowRE/unidown/master/tests/last_update_time.txt',␣
↪self._temp_dir, 'last_update_time.txt')
    with self._temp_dir.joinpath('last_update_time.txt').open(encoding='utf8') as␣
↪reader:
        return datetime.strptime(reader.read(), LinkItem.time_format)

def _load_default_options(self):
    super(Plugin, self)._load_default_options()
    if 'username' not in self._options:
        self._options['username'] = ''
```

So if the generate update time is newer, the application continues to `_create_download_links`. In the example we just download a json with preconfigured values and create the `LinkItemDict` from it.

```
class Plugin(APlugin):
    _info = PluginInfo('test', '0.1.0', 'raw.githubusercontent.com')

def __init__(self, settings: Settings, options: Dict[str, Any] = None):
    super().__init__(settings, options)
    self._username: str = self._options['username']

def _create_last_update_time(self) -> datetime:
    self.download_as_file('/IceflowRE/unidown/master/tests/last_update_time.txt',␣
↪self._temp_dir, 'last_update_time.txt')
    with self._temp_dir.joinpath('last_update_time.txt').open(encoding='utf8') as␣
↪reader:
        return datetime.strptime(reader.read(), LinkItem.time_format)

def _create_download_data(self) -> LinkItemDict:
    self.download_as_file('/IceflowRE/unidown/master/tests/item_dict.json', self._
↪temp_dir, 'item_dict.json')
    with self._temp_dir.joinpath('item_dict.json').open(encoding='utf8') as reader:
        data = json.loads(reader.read())
    result = LinkItemDict()
    for link, item in data.items():
        result[link] = LinkItem(item['name'], datetime.strptime(item['time'],␣
↪LinkItem.time_format))
    return result

def _load_default_options(self):
    super(Plugin, self)._load_default_options()
    if 'username' not in self._options:
        self._options['username'] = ''
```

Final step includes that we create a python package out of the generate file.

```
unidown_test/
├── __init__.py
├── plugin.py  «the plugin file»
└── setup.py
```

The most important part is the entry point, otherwise unidown cannot recognize it.

`'unidown.plugin': "test = unidown_test.plugin:Plugin"` Where as `test` is the name of the

plugin and after that the import path the plugin class.

Listing 1: setup.py

```
from setuptools import find_packages, setup

setup(
    name="unidown_test",
    version="0.1.0",
    description="Test plugin for unidown.",
    author="Iceflower S",
    author_email="iceflower@gmx.de",
    license='MIT',
    packages=find_packages(include=['unidown_test', 'unidown_test.*']),
    python_requires='>=3.7',
    install_requires=[
        'unidown',
    ],
    entry_points={
        'unidown.plugin': "test = unidown_test.plugin:Plugin"
    },
)
```

### 2.2.2 Advanced

The advanced part will show how to use a custom savestate.

First of all we have to create the custom savestate. It is required to subclass from *unidown.plugin.savestate.SaveState*.

```
from unidown.plugin.savestate import SaveState

class MySaveState(SaveState):
```

In our example we want to permanently store a username.

```
class MySaveState(SaveState):

    def __init__(self, plugin_info: PluginInfo, last_update: datetime, link_items:␣
↪LinkItemDict, username: str = ''):
        super().__init__(plugin_info, last_update, link_items)
        self.username: str = username
```

Furthermore to get it loaded from a custom json file we have to override `from_json`, the same goes with saving with `to_json`.

```
class MySaveState(SaveState):

    def __init__(self, plugin_info: PluginInfo, last_update: datetime, link_items:␣
↪LinkItemDict, username: str = ''):
        super().__init__(plugin_info, last_update, link_items)
        self.username: str = username

    @classmethod
    def from_json(cls, data: dict) -> SaveState:
        savestate = super(MySaveState, cls).from_json(data)
```

(continues on next page)

(continued from previous page)

```
        if 'username' in data:
            savestate.username = data['username']
        return savestate

    def to_json(self) -> dict:
        data = super().to_json()
        data['username'] = self.username
        return data
```

Additionally it's required to provide an own `upgrade` method, in case the format changes and as the super method may erase your own set variables while upgrading.

```
class MySaveState(SaveState):

    def __init__(self, plugin_info: PluginInfo, last_update: datetime, link_items:␣
↪LinkItemDict, username: str = ''):
        super().__init__(plugin_info, last_update, link_items)
        self.username: str = username

    @classmethod
    def from_json(cls, data: dict) -> SaveState:
        savestate = super(MySaveState, cls).from_json(data)
        if 'username' in data:
            savestate.username = data['username']
        return savestate

    def to_json(self) -> dict:
        data = super().to_json()
        data['username'] = self.username
            return data

    def upgrade(self) -> SaveState:
        new_savestate = super(MySaveState, self).upgrade()
        new_savestate.username = self.username
        return new_savestate
```

The next step is to register your own custom savestate in the plugin, so it will be used, by setting `_savestate_cls`.

```
from unidown_test.savestate import MySaveState

class Plugin(APlugin):
    _info = PluginInfo('test', '0.1.0', 'raw.githubusercontent.com')
    _savestate_cls = MySaveState
```

To get the username loaded into your plugin, after the savestate was loaded override `load_savestate`.

```
from unidown_test.savestate import MySaveState

class Plugin(APlugin):
    _info = PluginInfo('test', '0.1.0', 'raw.githubusercontent.com')
    _savestate_cls = MySaveState

# ... all other stuff in between

def load_savestate(self):
    super(Plugin, self).load_savestate()
```

(continues on next page)

(continued from previous page)

```
    # do not override set username by options
    if self._username == '':
        self._username = self.savestate.username
```

The last step is to hook into updating the savestate. To get the current username saved into a new savestate.

```
from unidown_test.savestate import MySaveState

class Plugin(APlugin):
    _info = PluginInfo('test', '0.1.0', 'raw.githubusercontent.com')
    _savestate_cls = MySaveState

# ... all other stuff in between

def load_savestate(self):
    super(Plugin, self).load_savestate()
    # do not override set username by options
    if self._username == '':
        self._username = self.savestate.username

def update_savestate(self, new_items: LinkItemDict):
    super(Plugin, self).update_savestate(new_items)
    self._savestate.username = self._username
```

CHAPTER

# THREE

# DEVELOPMENT

## 3.1 CI building

### 3.1.1 Travis CI Environment Variables

**PYPI_USER** PYPI and Test PYPI User.

**PYPI_PASSWORD** PYPI Password for the user above.

**PYPI_TEST_PASSWORD** Test PYPI password for the user.

**CODACY_PROJECT_TOKEN** Codacy token for the project.

## 3.2 Functionality

1. Get plugin from the given name
2. Get the last overall update time
3. Load the savestate
4. Compare last update time with the one from the savestate
5. Get the download links
6. Compare received links and their times with the savestate
7. Clean up names, to eliminate duplicated
8. Download new and newer links
9. Check downloaded data
10. Update savestate
11. Save new savestate to file

CHAPTER

FOUR

# INTERNALS

Reference material.

## 4.1 unidown

### 4.1.1 unidown.main

Entry into the program.

**class** unidown.main.**PluginListAction**(*option_strings*, *dest*, ***kwargs*)
Lists all plugins which are available. Extension for the argparser.

unidown.main.**main**(*argv=None*)
Entry point into the program. Gets the arguments from the console and proceed them with ArgumentParser. Returns if its success successful 0 else 1.

### 4.1.2 unidown.static_data

Static and important data which is needed by the program and do not need any third party libraries. (this is important because it is used inside the setup.py | **do not edit**.

unidown.static_data.**AUTHOR = 'Iceflower S'**
author

unidown.static_data.**AUTHOR_EMAIL = 'iceflower@gmx.de'**
author email

unidown.static_data.**AUTHOR_GITHUB = 'https://github.com/IceflowRE'**
author github link

unidown.static_data.**DESCRIPTION = 'Universal downloader, a modular extensible downloader wh**
short description

unidown.static_data.**LONG_NAME = 'Universal Downloader'**
long name of this program

unidown.static_data.**NAME = 'unidown'**
name of this program

unidown.static_data.**PROJECT_URL = 'https://github.com/IceflowRE/unidown'**
project url

unidown.static_data.**PYPI_JSON_URL = 'https://pypi.org/pypi/unidown/json'**
url to the unidown pypi json

unidown.static_data.**VERSION = '2.0.2'**
version in PEP440 format

### 4.1.3 unidown.tools

Different tools.

unidown.tools.**print_plugin_list**(*plugins*)
Prints all registered plugins and checks if they can be loaded or not.

**Parameters** **plugins** (Dict[str, EntryPoint]) – plugins name to entrypoint

unidown.tools.**unlink_dir_rec**(*path*)
Delete a folder recursive.

**Parameters** **path** (Path) – folder to deleted

## 4.2 unidown.core

### 4.2.1 unidown.core.manager

Manager of the whole program, contains the most important functions as well as the download routine.

unidown.core.manager.**check_update**()
Check for app updates and print/log them.

unidown.core.manager.**download_from_plugin**(*plugin*)
Download routine.

1. Get plugin from the given name
2. Get the last overall update time
3. Load the savestate
4. Compare last update time with the one from the savestate
5. Get the download links
6. Compare received links and their times with the savestate
7. Clean up names, to eliminate duplicated
8. Download new and newer links
9. Check downloaded data
10. Update savestate
11. Save new savestate to file

**Parameters** **plugin** (*APlugin*) – plugin

unidown.core.manager.**get_options**(*options*)
Convert the option list to a dictionary where the key is the option and the value is the related option. Is called in the init.

**Parameters** **options** (List[List[str]]) – options given to the plugin.

**Return type** Dict[str, Any]

**Returns** dictionary which contains the option key and values

unidown.core.manager.**init_logging**(*settings*)
Initialize the _downloader.

**Parameters** **settings** (*Settings*) – settings

unidown.core.manager.**run**(*settings*, *plugin_name*, *raw_options*)
Run a plugin so use the download routine and clean up after.

**Parameters**

- **settings** (*Settings*) – settings to use
- **plugin_name** (str) – name of plugin
- **raw_options** (List[List[str]]) – parameters which will be send to the plugin initialization

**Return type** *PluginState*

**Returns** ending state

unidown.core.manager.**shutdown**()
Close and exit important things.

### 4.2.2 unidown.core.plugin_state

**class** unidown.core.plugin_state.**PluginState**(*value*)
State of a plugin, after it ended or was not found.

**EndSuccess = 0**
successfully end

**LoadCrash = 3**
raised an exception while loading/ initializing

**NotFound = 4**
plugin was not found

**RunCrash = 2**
raised an exception but ~unidown.plugin.exceptions.PluginException

**RunFail = 1**
raised an ~unidown.plugin.exceptions.PluginException

### 4.2.3 unidown.core.settings

**class** unidown.core.settings.**Settings**(*root_dir=None*, *log_file=None*, *log_level='INFO'*)

**Variables**

- **_root_dir** – root path
- ***temp_dir*** – temporary main path, here are the sub folders for every plugin
- ***download_dir*** – download main path, here are the sub folders for every plugin
- ***savestate_dir*** – savestates main path, here are the sub folders for every plugin
- ***log_file*** – log file of the program

- **available_plugins** – available plugins which are found at starting the program, name -> EntryPoint
- **using_cores** – how many _cores should be used
- **_log_level** – log level
- **_disable_tqdm** – if the console progress bar is disabled

**Parameters**

- **root_dir** (Optional[Path]) – root dir
- **log_file** (Optional[Path]) – log file

**check_dirs**()
Check the directories if they exist.

**Raises** **FileExistsError** – if a file exists but is not a directory

**property cores**
Plain getter.

**Return type** int

**property disable_tqdm**
Plain getter.

**Return type** bool

**property download_dir**
Plain getter.

**Return type** Path

**property log_file**
Plain getter.

**Return type** Path

**property log_level**
Plain getter.

**Return type** str

**mkdir**()
Create all base directories.

**property root_dir**
Plain getter.

**Return type** Path

**property savestate_dir**
Plain getter.

**Return type** Path

**property temp_dir**
Plain getter.

**Return type** Path

### 4.2.4 unidown.core.updater

Things needed for checking for updates.

unidown.core.updater.**check_for_app_updates**()
Check for updates.

**Return type** bool

**Returns** is update available

unidown.core.updater.**get_newest_app_version**()
Download the version tag from remote.

**Return type** Version

**Returns** version from remote

## 4.3 unidown.plugin

### 4.3.1 unidown.plugin.a_plugin

**class** unidown.plugin.a_plugin.**APlugin**(*settings*, *options=None*)
Abstract class of a plugin. Provides all needed variables and methods.

**Parameters** **options** (Optional[Dict[str, Any]]) – parameters which can included optional parameters

**Raises** ***PluginException*** – can not create default plugin paths

**Variables**

- **_info** – information about the plugin
- ***_savestate_cls*** – savestate class to use
- **_disable_tqdm** – if the tqdm progressbar should be disabled | **do not edit**
- **_log** – use this for logging | **do not edit**
- **_simul_downloads** – number of simultaneous downloads
- **_temp_dir** – path where the plugin can place all temporary data | **do not edit**
- **_download_dir** – general download path of the plugin | **do not edit**
- **_savestate_file** – file which contains the latest savestate of the plugin | **do not edit**
- **_last_update** – latest update time of the referencing data | **do not edit**
- **_unit** – the thing which should be downloaded, may be displayed in the progress bar
- **_download_data** – referencing data | **do not edit**
- **_downloader** – downloader which will download the data | **do not edit**
- **_savestate** – savestate of the plugin
- **_options** – options which the plugin uses internal, should be used for the given options at init

**abstract** **_create_download_data**()
Get the download links in a specific format. **Has to be implemented inside Plugins.**

**Raises** **NotImplementedError** – abstract method

**Return type** *LinkItemDict*

**abstract _create_last_update_time**()
Get the newest update time from the referencing data. **Has to be implemented inside Plugins.**

**Raises** **NotImplementedError** – abstract method

**Return type** datetime

**_load_default_options**()
Loads default options if they were not passed at creation.

**_savestate_cls**
alias of *unidown.plugin.savestate.SaveState*

**check_download**(*link_item_dict*, *folder*, *log=False*)
Check if the download of the given dict was successful. No proving if the content of the file is correct too.

**Parameters**

- **link_item_dict** (*LinkItemDict*) – dict which to check
- **folder** (Path) – folder where the downloads are saved
- **log** (bool) – if the lost items should be logged

**Return type** Tuple[*LinkItemDict*, *LinkItemDict*]

**Returns** succeeded and failed

**clean_up**()
Default clean up for a module. Deletes _temp_dir.

**download**(*link_items*, *folder*, *desc*, *unit*)

> **Warning:** The parameters may change in future versions. (e.g. change order and accept another host)

Download the given LinkItem dict from the plugins host, to the given path. Proceeded with multiple connections _simul_downloads. After *check_download()* is recommend.

This function don't use an internal *link_item_dict*, *delay* or *folder* directly set in options or instance vars, because it can be used aside of the normal download routine inside the plugin itself for own things. As of this it still needs access to the logger, so a staticmethod is not possible.

**Parameters**

- **link_items** (*LinkItemDict*) – data which gets downloaded
- **folder** (Path) – target download folder
- **desc** (str) – description of the progressbar
- **unit** (str) – unit of the download, shown in the progressbar

**download_as_file**(*url*, *target_file*, *delay=0*)
Download the given url to the given target folder.

**Parameters**

- **url** (str) – link
- **target_file** (Path) – target file

- **delay** (float) – after download wait this much seconds

**Return type** str

**Returns** url

**Raises** **HTTPError** – if the connection has an error

**property download_data**
Plain getter.

**Return type** *LinkItemDict*

**property download_dir**
Plain getter.

**Return type** Path

**static get_plugins**()
Get all available plugins for unidown.

**Return type** Dict[str, EntryPoint]

**Returns** plugin name list

**property host**
Plain getter.

**Return type** str

**property info**
Plain getter.

**Return type** *PluginInfo*

**property last_update**
Plain getter.

**Return type** datetime

**load_savestate**()
Load the save of the plugin.

**Raises**

- ***PluginException*** – broken savestate json
- ***PluginException*** – different savestate versions
- ***PluginException*** – different plugin versions
- ***PluginException*** – different plugin names
- ***PluginException*** – could not parse the json

**property log**
Plain getter.

**Return type** Logger

**property name**
Plain getter.

**Return type** str

**property options**
Plain getter.

**Return type** Dict[str, Any]

**save_savestate**()
Save meta data about the downloaded things and the plugin to file.

**property savestate**
Plain getter.

**property simul_downloads**
Plain getter.

**Return type** int

**property temp_dir**
Plain getter.

**Return type** Path

**property unit**
Plain getter.

**Return type** str

**update_download_data**()
Update the download links. Calls *_create_download_data()*.

**update_last_update**()
Call this to update the latest update time. Calls *_create_last_update_time()*.

**update_savestate**(*new_items*)
Update savestate.

**Parameters** **new_items** (*LinkItemDict*) – new items

**property version**
Plain getter.

**Return type** Version

### 4.3.2 unidown.plugin.exceptions

Default exceptions of plugins.

**exception** unidown.plugin.exceptions.**PluginException**(*msg=''*)
Base class for exceptions in a plugin. If catching this, it implicit that the plugin is unable to work further.

**Parameters** **msg** (str) – message

**Variables** **msg** – exception message

### 4.3.3 unidown.plugin.link_item

**class** unidown.plugin.link_item.**LinkItem**(*name*, *time*)
Item which represents the data, who need to be downloaded. Has a name and an update time.

**Parameters**

- **name** (str) – name
- **time** (datetime) – update time

**Raises**

- **ValueError** – name cannot be empty or None
- **ValueError** – time cannot be empty or None

**Variables**

- **time_format** – time format to use
- **_name** – name of the item
- **_time** – time of the item

**classmethod from_json**(*data*)
Constructor from json dict.

**Parameters** **data** (dict) – json data as dict

**Return type** *LinkItem*

**Returns** the LinkItem

**Raises** **ValueError** – missing parameter

**property name**
Plain getter.

**Return type** str

**property time**
Plain getter.

**Return type** datetime

**to_json**()
Create json data.

**Return type** dict

**Returns** json dictionary

### 4.3.4 unidown.plugin.link_item_dict

**class** unidown.plugin.link_item_dict.**LinkItemDict**
LinkItem dictionary, acts as a wrapper for special methods and functions.

**actualize**(*new_data*, *log=None*)
Actualize dictionary like an ~dict.update does. If a logger is passed it will log updated items, **not** new one.

**Parameters**

- **new_data** (*LinkItemDict*) – the data used for updating
- **log** (Optional[Logger]) – logger

**clean_up_names**()
Rename duplicated names with an additional _r.

**static get_new_items**(*old_data*, *new_data*, *disable_tqdm=False*)
Get the new items which are not existing or are newer as in the old data set.

**Parameters**

- **old_data** (*LinkItemDict*) – old data
- **new_data** (*LinkItemDict*) – new data
- **disable_tqdm** (bool) – disables tqdm progressbar

**Return type** *LinkItemDict*

**Returns** new and updated link items

### 4.3.5 unidown.plugin.plugin_info

**class** unidown.plugin.plugin_info.**PluginInfo**(*name*, *version*, *host*)

Information about the module. Those information will be saved into the save files as well.

**Parameters**

- **name** (str) – the name of the plugin
- **version** (str) – version, PEP440 conform
- **host** (str) – host url of the main data

**Raises**

- **ValueError** – name is empty
- **ValueError** – host is empty
- **InvalidVersion** – version is not PEP440 conform

**Variables**

- ***name*** – name of the plugin
- ***host*** – host url of the main data
- ***version*** – plugin version

**classmethod from_json**(*data*)

Constructor from json dict.

**Parameters** **data** (dict) – json data as dict

**Return type** *PluginInfo*

**Returns** the plugin info

**Raises**

- **ValueError** – name is missing
- **ValueError** – version is missing
- **ValueError** – host is missing

**to_json**()

Create json data.

**Return type** dict

**Returns** json dictionary

### 4.3.6 unidown.plugin.savestate

**class** unidown.plugin.savestate.**SaveState**(*plugin_info*, *last_update*, *link_items*, *version=<Version('1')>*)

Savestate of a plugin.

**Parameters**

- **version** (Version) – savestate version
- **plugin_info** (*PluginInfo*) – plugin info
- **last_update** (datetime) – last udpate time of the referenced data
- **link_items** (*LinkItemDict*) – data
- **version** – savestate version

**Variables**

- **time_format** – time format to use
- ***version*** – savestate version
- ***plugin_info*** – plugin info
- ***last_update*** – newest udpate time
- **link_items** – data

**classmethod from_json**(*data*)

**Parameters** **data** (dict) – json data as dict

**Return type** *SaveState*

**Returns** the SaveState

**Raises**

- **ValueError** – version of SaveState does not exist or is empty
- **InvalidVersion** – version is not PEP440 conform

**to_json**()

Create json data.

**Return type** dict

**Returns** json dictionary

**upgrade**()

Upgrading current savestate to the latest savestate version.

**Return type** *SaveState*

**Returns** upgraded savestate

# PYTHON MODULE INDEX

## u

# INDEX

## Symbols



## A



## C



## D

## S



## T



## U



## V