pycliarr

Release 1.0.22

Sep 24, 2022

Userguide

CHAPTER 1

What's pycliarr

Python client for radarr and sonarr apis. The package provides python client and CLI to use in command line.

Documentation homepage

CHAPTER 2

Usage

Sonarr CLI

```
pyvenv/bin/pycliarr -t "http://192.168.0.199:8989" -k
 ↪"2ac2d8f667524da3bx1849e81dba5a84" -d sonarr get -i 65
pyvenv/bin/pycliarr -t "http://192.168.0.199:8989" -k
 ↪"2ac2d8f667524da3bax849e81dba5a84" -d sonarr add -t "the walking dead"
```

Radarr CLI

```
pyvenv/bin/pycliarr -t "http://192.168.0.199:7878" -k
 ↪"2ac2d8f667524da3bx1849e81dba5a84" -d radarr get -i 65
pyvenv/bin/pycliarr -t "http://192.168.0.199:7878" -k
 ↪"2ac2d8f667524da3bax849e81dba5a84" -d radarr add -t "wonder woman"
```

Using radarr client

```
from pycliarr.api import RadarrCli
radarr_cli = RadarrCli('http://192.168.0.199:7878', '5f5e32qf3ff8463e9f3d2388af0fd3e8
 ↪')
radarr_cli.add_movie(imdb_id="tt1234", quality=1)
movie = radarr_cli.get_movie(12)
print(movie.title)
```

Using sonarr client

```
from pycliarr.api import SonarrCli
sonarr_cli = SonarrCli('http://192.168.0.199:8989', '2ac2d8f667524da3bx1849e81dba5a84
 ↪')
sonarr_cli.add_serie(imdb_id="tt1234", quality=1)
serie = sonarr_cli.get_serie(12)
print(serie.title)
```

CHAPTER 3

CLI help

Clients:

```
pyvenv/bin/pycliarr --help
PyCliarr version 1.0.21
usage: pycliarr [-h] --host HOST --api-key API_KEY [--user USER] [--password␣
↪PASSWORD] [--debug] {sonarr,radarr} ...

Radarr/Sonarr client

positional arguments:
  {sonarr,radarr}
    sonarr              use sonarr client
    radarr              use radarr client

optional arguments:
  -h, --help            show this help message and exit
  --host HOST, -t HOST  Host url, e.g 'http://192.168.0.1'
  --api-key API_KEY, -k API_KEY
                        API key, e.g '5f5e32xf3ff8463d9f1d2u88ef0fd3e8'
  --user USER, -u USER  Username if using basic authentication
  --password PASSWORD, -p PASSWORD
                        Password if using basic authentication
  --debug, -d           Enable debug logging
```

Radarr CLI:

```
pyvenv/bin/pycliarr radarr --help
PyCliarr version 1.0.21
usage: pycliarr radarr [-h]
                       {get,delete,add,edit,refresh,rescan,profiles,system-status,
↪disk-space,queue,calendar,delete-queue,wanted,status,blocklist,delete-blocklist,
↪notification,delete-notification,put-notification,tag,tag-detail,delete-tag,edit-
↪tag,create-tag,tag-items,exclusion,delete-exclusion,create-exclusion,search-missing,
↪root-folders}
```

(continues on next page)

(continued from previous page)

```
                        ...

positional arguments:
  {get,delete,add,edit,refresh,rescan,profiles,system-status,disk-space,queue,
↪calendar,delete-queue,wanted,status,blocklist,delete-blocklist,notification,delete-
↪notification,put-notification,tag,tag-detail,delete-tag,edit-tag,create-tag,tag-
↪items,exclusion,delete-exclusion,create-exclusion,search-missing,root-folders}
    get                 Get info on a of movie
    delete              Delete a movie
    add                 Add a movie from the imdb/tmdb id, or look for keywords
    edit                Push an updated item to the movie library
    refresh             Refresh movies
    rescan              Rescan movies
    profiles            Get list of quality profiles
    system-status       Get system status
    disk-space          Get disk space
    queue               Get current downloading queue
    calendar            Get events from calendar
    delete-queue        Get list of quality profiles
    wanted              List wanted/missing
    status              Get status of 1 or all currently running commands
    blocklist           Get blocklisted items
    delete-blocklist    Get list of quality profiles
    notification        Get notification(s)
    delete-notification
                        Delete the specified notification or all
    put-notification    Create the specified notification
    tag                 Get tag(s)
    tag-detail          Get tag(s) details
    delete-tag          Delete the specified tag
    edit-tag            Edit the specified tag
    create-tag          Create the specified tag
    tag-items           List items with specifed tag
    exclusion           Get exclusion(s)
    delete-exclusion    Delete the specified exclusion
    create-exclusion    Create the specified exclusion
    search-missing      Search missing movies
    root-folders        Get root folder list

optional arguments:
  -h, --help            show this help message and exit
```

Sonarr CLI:

```
pyvenv/bin/pycliarr sonarr --help
PyCliarr version 1.0.21
usage: pycliarr sonarr [-h]
                       {get,delete,add,refresh,rescan,get-episode,get-episode-file,
↪delete-episode-file,profiles,system-status,disk-space,queue,calendar,delete-queue,
↪wanted,status,blocklist,delete-blocklist,notification,delete-notification,put-
↪notification,tag,tag-detail,delete-tag,edit-tag,create-tag,tag-items,exclusion,
↪delete-exclusion,create-exclusion,search-missing,root-folders}
                       ...

positional arguments:
  {get,delete,add,refresh,rescan,get-episode,get-episode-file,delete-episode-file,
↪profiles,system-status,disk-space,queue,calendar,delete-queue,wanted,status,
↪blocklist,delete-blocklist,notification,delete-notification,put-notification,tag,
↪tag-detail,delete-tag,edit-tag,create-tag,tag-items,exclusion,delete-exclusion,
↪create-exclusion,search-missing,root-folders}
```

(continues on next page)

(continued from previous page)

```
    get                 Get info on a of serie
    delete              Delete a serie
    add                 Add a serie from the tvdb id, or look for keywords
    refresh             Refresh series
    rescan              Rescan series
    get-episode         Get info on an episode
    get-episode-file    Get info on an episode file
    delete-episode-file
                        Get info on a of serie
    profiles            Get list of quality profiles
    system-status       Get system status
    disk-space          Get disk space
    queue               Get current downloading queue
    calendar            Get events from calendar
    delete-queue        Get list of quality profiles
    wanted              List wanted/missing
    status              Get status of 1 or all currently running commands
    blocklist           Get blocklisted items
    delete-blocklist    Get list of quality profiles
    notification        Get notification(s)
    delete-notification
                        Delete the specified notification or all
    put-notification    Create the specified notification
    tag                 Get tag(s)
    tag-detail          Get tag(s) details
    delete-tag          Delete the specified tag
    edit-tag            Edit the specified tag
    create-tag          Create the specified tag
    tag-items           List items with specifed tag
    exclusion           Get exclusion(s)
    delete-exclusion    Delete the specified exclusion
    create-exclusion    Create the specified exclusion
    search-missing      Search missing episods
    root-folders        Get root folder list

optional arguments:
  -h, --help            show this help message and exit
```

CHAPTER 4

Installation

From pip:

```
pip pycliarr
```

From git:

```
git clone https://github.com/vche/pycliarr.git
pip install -e .
```

CHAPTER 5

Development

5.1 Installing sources projects

Get the project and create the virtual env:

```
git clone https://github.com/vche/pycliarr.git
virtualenv pyvenv
. pyvenv/bin/activate
pip install -e .
```

Note: Entry points will be installed in pyvenv/bin, libs with pyvenv libs

5.2 Run tests

```
pip install tox
tox
```

If mypy fails due to missing import stubs:

```
.tox/checkers/bin/mypy --install-types
```

5.3 Generate documentation:

```
pip install sphinx sphinx_rtd_theme m2r2
./setup.py doc
```

In case new classes/modules are added, update the autodoc list:

```
rm  docs/sphinx_conf/source/*
sphinx-apidoc -f -o docs/sphinx_conf/source/ src/pycliarr --separate
```

CHAPTER 6

Changelog

6.1 master

6.2 v1.0.22

Date Sept 24, 2022

6.2.1 Fix

- Fix missing raw option and support for older root folder requests (without space)

6.3 v1.0.21

Date Sept 23, 2022

6.3.1 Fix

- Fix unit tests for older python versions, mypy issues and add workflows

6.4 v1.0.20

Date Sept 23, 2022

6.4.1 New

- **Issue #30:**
  - Add command line option so list root folders.
  - Add command line option to specify or select a root folder to use when adding movie/show.
  - Update add methods to allow specifying root id.
  - Allow specifying default root folder when creating client

6.5 v1.0.19

Date Jan 22, 2022

6.5.1 New

- Add commands to search missing movies and episodes
- Add movie/serie edit commands

6.5.2 Fix

- Missing qet queue parameters
- Missing int cast in build root path
- Missing root folder argument in build path

6.6 v1.0.18

Date Nov 29, 2021

6.6.1 Fix

- fix issue with root paths in radnarr with api v3

6.7 v1.0.17

Date Nov 26, 2021

6.7.1 Fix

- fix radarr lookup url
- fix sonarr add serie new api with language id

6.7.2 New

- add import list inclusion api

6.8 v1.0.16

Date Nov 25, 2021

6.8.1 Fix

- delete serie with exclusion list

6.8.2 New

- Fully moved to api v3 on both sonarr and radarr
- Added blocklist commands
- Added notification commands
- Added tag commands

6.9 v1.0.15

Date Nov 23, 2021

6.9.1 Fix

- Fix delete movie exclusion option for api v3

6.10 v1.0.14

Date June 15, 2021

6.10.1 Fix

- Remove unsupported chars from movie/serie paths depending on the platform

6.11 v1.0.13

Date May 23, 2021

6.11.1 New

- Add option to specify folder path in add_movie and add_serie
- Default folder path builders
- Update default movie folder with release year to match radarr gui default

6.11.2 Fix

- Support for several root folders in get_root_folder()

6.12 v1.0.12

Date May 16, 2021

6.12.1 Fix

- Fix issue with default values for dates

6.13 v1.0.11

Date May 16, 2021

6.13.1 Fix

- Fix wrong url format with delete queue commands

6.14 v1.0.10

Date May 14, 2021

6.14.1 Fix

- Remove debug log

6.15 v1.0.9

Date May 13, 2021

6.15.1 Fix

- Add missing files to radarr item
- Fix issue when a single item is returned as lookup results

6.16 v1.0.8

Date May 9, 2021

6.16.1 New

- Issue with delete requests parameters sent as data instead of url parameters

6.16.2 New

- Add season folder creation option to sonarr

6.17 v1.0.7

Date May 3, 2021

6.17.1 New

- Added optional selection of seaons to monitor in sonarr.add_serie(), (use case from https://github.com/marc0janssen/pixlovarr)

6.18 v1.0.6

Date Jan 19, 2021

6.18.1 Fix

- Fix bug when servers return an array of 1 element

6.19 v1.0.5

Date Dec 18, 2020

6.19.1 New

- Add raw server response in server exception
- Add classes imports to api module

6.19.2 Fix

- Radarr quality profile parsing issue in CLI aith api v3
- Cleanup debug logs

6.20 v1.0.4

Date Dec 17, 2020

6.20.1 New

- Added cli status command
- Use radarr api v3

6.21 v1.0.3

Date Aug 30, 2020

6.21.1 Fix

- Re release of 1.0.2 with updated doc

6.22 v1.0.2

Date Aug 28, 2020

6.22.1 Fix

- Fix issue when adding using tmdb/imdb/tvdb id

6.23 v1.0.1

Date Aug 26, 2020

6.23.1 New

- Full unit tests coverage
- Available in pip
- Full command set

6.24 v0.0.1

Date Aug 23, 2020

6.24.1 New

- Initial version with sonarr and radarr clients

CHAPTER 7

pycliarr

7.1 pycliarr package

7.1.1 Subpackages

pycliarr.api package

Submodules

pycliarr.api.base_api module

class pycliarr.api.base_api.**BaseCliApi**(*host_url: str*, *api_key: str*, *username: Optional[str] = None*, *password: Optional[str] = None*)

Bases: object

Low level base API client class.

Provides basic requests access (put/get/post/delete) to an API, handling api key and basic authentication

api_key

close() → None
Close session with the endpoint.

host_url

request(*method: str*, *path: str*, *url_params: Optional[Dict[str, Any]] = None*, *json_data: Union[Dict[str, Any], List[Dict[str, Any]], None] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Send a request to the host API

Parameters

- **method** –
- **path** (*str*) – host endpoint path. Must start with a '/'. e.g. /api/queue

- **url_params** (*Optional[Dict[str, Any]]*) – Optional list of query parameters. e.g. {'term': 'some keyword'}
- **json_data** (*Optional[json_data]*) – Optional JSON data to send

Returns Response object form requests.

Return type requests.models.Response

request_delete(*path: str*, *url_params: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Shortcut for request withe method=delete.

request_get(*path: str*, *url_params: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Shortcut for request withe method=get.

request_post(*path: str, json_data: Union[Dict[str, Any], List[Dict[str, Any]], None] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Shortcut for request withe method=post.

request_put(*path: str, json_data: Union[Dict[str, Any], List[Dict[str, Any]], None] = None, url_params: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Shortcut for request withe method=put.

to_path(*basename: str*) → pathlib.Path
Remove invalid chars from a file/directory name depending on the platform.

class pycliarr.api.base_api.**BaseCliApiItem**(***kwargs*)
Bases: object

Generic handling of an item based on a dict representation.

Items can be build specifying a list of parameters, a dict, or a json string. All fields are directly accessible as attributes.

This is especially usedul by clients to directly convert or create items received or to send by BaseCliApi subclasses

add_attribute(*name: str*, *value: Any*) → None

classmethod from_dict(*dict_data: Dict[Any, Any]*) → BaseItemClass
Build an item and populate it based on the given dictionnary.

classmethod from_json(*json_data: str*) → BaseItemClass
Build an item and populate it based on json data.

to_dict() → Dict[Any, Any]

to_json() → str

pycliarr.api.base_media module

class pycliarr.api.base_media.**BaseCliMediaApi**(**args*, *default_root_folder_id: int = 0*, ***kwargs*)
Bases: *pycliarr.api.base_api.BaseCliApi*

Base class for media based API.

Implement behavior common to media based apis (e.g. sonarr, radarr)

add_item(*json_data: Union[Dict[str, Any], List[Dict[str, Any]]]*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Adds a new item to collection

Parameters **json_data** – Dict representation of the item to add

Returns json response

api_url_base = '/api/v3'

api_url_blocklist = '/api/v3/blocklist'

api_url_calendar = '/api/v3/calendar'

api_url_command = '/api/v3/command'

api_url_diskspace = '/api/v3/diskspace'

api_url_exclusions = '/api/v3/importlistexclusion'

api_url_history = '/api/v3/history/'

api_url_item = '/api/v3/item'

api_url_itemlookup = '/api/v3/item/lookup'

api_url_language_profile = '/api/v3/languageProfile'

api_url_log = '/api/v3/log'

api_url_notification = '/api/v3/notification'

api_url_profile = '/api/v3/qualityProfile'

api_url_queue = '/api/v3/queue'

api_url_rootfolder = '/api/v3/rootfolder'

api_url_systembackup = '/api/v3/system/backup'

api_url_systemstatus = '/api/v3/system/status'

api_url_tag = '/api/v3/tag'

api_url_wanted_missing = '/api/v3/wanted/missing'

build_item_path(*title: str*, *root_folder_id: int = 0*) → pathlib.Path
Build an item folder path using the root folder specified. :param title: Title to add to root path. All invalid characters are removed :type title: str :param root_folder_id: Id of the root folder (can be retrieved with get_root_folder()). :type root_folder_id: int :param If the id is not found or not specified, the default root folder id will be used.: :param If 0, the first root folder in the list is used.:

Returns: Full path of the serie in the format <root path>/<serie name>

create_tag(*value: str*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Create the specified tag

Parameters

- **item_id** (*int*) – id of the tag to edit
- **value** (*str*) – Tag label

Returns json response

default_root_folder_id

delete_blocklist(*item_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Remove the specified item from the blocklist, or all items if none specified

Parameters **item_id** (*int*) – Item to delete, None to delete all items

Returns json response

delete_exclusion(*item_id: int*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Remove the specified exclusions

Parameters **item_id** (*int*) – id of the exclusions to delete

Returns json response

delete_item(*item_id: int*, *delete_files: bool = True*, *options: Dict[str, Any] = {}*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Delete the item with the given ID

Parameters

- **item_id** (*int*) – Item to delete
- **delete_files** (*bool*) – Optional. Also delete files. Default is False
- **options** (*Dict[str, Any]*) – Optionally specify additional options

Returns json response

delete_notification(*item_id: int*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Remove the specified item from the blocklist, or all items if none specified

Parameters **item_id** (*int*) – id of the notification to delete

Returns json response

delete_queue(*item_id: int*, *blacklist: Optional[bool] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Delete an item from the queue and download client. Optionally blacklist item after deletion.

Parameters

- **item_id** (*int*) – Item to delete
- **blacklist** (*Optional[bool]*) – Optionally blacklist the item

Returns json response

delete_tag(*item_id: int*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Remove the specified tag

Parameters **item_id** (*int*) – id of the notification to delete

Returns json response

edit_item(*json_data: Union[Dict[str, Any], List[Dict[str, Any]]]*, *url_params: Optional[Dict[str, Any]] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Edit an item from the collection

Parameters **json_data** – Dict representation of the item to add

Returns json response

edit_tag(*item_id: int*, *value: str*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Edit the specified tag

Parameters

- **item_id** (*int*) – id of the tag to edit
- **value** (*str*) – Tag label

Returns json response

get_backup() → Union[Dict[str, Any], List[Dict[str, Any]]]
Return the backups as json

get_blocklist(*page: int = 1*, *sort_key: str = 'date'*, *page_size: int = 20*, *sort_dir: str = 'descending'*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get blocklisted releases

Parameters

- **page** (*int) - 1-indexed (1 default*) –
- **sort_key** (*string*) –
- **page_size** (*int*) – 20
- **sort_dir** (*string*) – descending

get_calendar(*start_date: Optional[datetime.datetime] = None*, *end_date: Optional[datetime.datetime] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Retrieve info about when items were/will be downloaded.

If start and end are not provided, retrieves movies airing today and tomorrow. :param start_date: Start date of events to retrieve :type start_date: Optional[datetime] :param end_date: End date of events to retrieve :type end_date: Optional[datetime]

Returns json response

get_command(*cid: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Query the status of a previously started command, or all currently running.

Parameters **cid** (*Optional[int]*) –

Returns json response

get_disk_space() → Union[Dict[str, Any], List[Dict[str, Any]]]
Retrieve info about the disk space on the server.

Returns json response

get_exclusion(*item_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get import list exclusions

Parameters **item_id** (*int*) – id of the exclusion to get, or None to get all of them

Returns json response

get_history(*page: int = 1*, *sort_key: str = 'date'*, *page_size: int = 10*, *sort_dir: str = 'asc'*, *options: Dict[str, Any] = {}*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get history (grabs/failures/completed)

Parameters

- **page** (*int) - 1-indexed (1 default*) –
- **sort_key** (*string*) –
- **page_size** (*int*) – 10
- **sort_dir** (*string*) – asc
- **options** (*Dict[str, Any]={}*) – Optional additional options

Returns json response

get_item(*item_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get specified item, or all if no id provided from server collection.

Parameters **item_id** (*Optional[int]*) –

Returns json response

get_language_profiles() → List[Dict[str, Any]]
Return the quality profiles

get_logs(*page: int = 1*, *sort_key: str = 'time'*, *page_size: int = 10*, *sort_dir: str = 'asc'*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get logs

Parameters

- **page** (*int) - 1-indexed (1 default*) –
- **sort_key** (*string*) –
- **page_size** (*int*) – 10
- **sort_dir** (*string*) – asc

Returns json response

get_notification(*item_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get specified notification or all if none specified

Parameters **item_id** (*int*) – id of the notification to get, or None to get all of them

Returns json response

get_quality_profiles() → List[Dict[str, Any]]
Return the quality profiles

get_queue(*page: int = 1*, *sort_key: str = 'progress'*, *page_size: int = 20*, *sort_dir: str = 'ascending'*, *include_unknown: bool = True*) → Union[Dict[str, Any], List[Dict[str, Any]]]

get_root_folder() → List[Dict[str, Any]]
Retrieve the server root folder.

Returns json response

get_system_status() → Union[Dict[str, Any], List[Dict[str, Any]]]
Return the System Status as json

get_tag(*item_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get specified tag or all if none specified

Parameters **item_id** (*int*) – id of the tag to get, or None to get all of them

Returns json response

get_tag_detail(*item_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get specified tag detail or all if none specified

Parameters **item_id** (*int*) – id of the tag to get, or None to get all of them

Returns json response

get_wanted(*page: int = 1*, *sort_key: str = 'airDateUtc'*, *page_size: int = 10*, *sort_dir: str = 'asc'*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get Wanted / Missing episodes

Parameters

- **sort_key** (*str*) – series.title or airDateUtc (default)
- **page** (*int*) – 1-indexed Default: 1
- **page_size** (*int*) – Default: 10

- **sort_dir** (*str*) – asc or desc - Default: asc

Returns json response

lookup_item(*term: str*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Search for items

Parameters **term** (*str*) – Lookup terms

Returns json response

put_notification(*item_id: int, notification_data: Union[Dict[str, Any], List[Dict[str, Any]]]*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Create the specified notification

Parameters

- **item_id** (*int*) – id of the notification to create
- **notification_data** (*json_data*) – Json dict describing the notification formated as in https://radarr.video/docs/api/#/Notification/put-notification-id

Returns json response

rename_files(*file_ids: List[int]*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Rename the list of files provided.

Parameters **file_ids** (*List[int]*) – List of ids of files to rename

Returns json response

sync_rss() → Union[Dict[str, Any], List[Dict[str, Any]]]
Perform an RSS sync with all enabled indexers.

Returns json response

pycliarr.api.exceptions module

exception pycliarr.api.exceptions.**CliArrError**
Bases: Exception

exception pycliarr.api.exceptions.**CliDecodeError**
Bases: *pycliarr.api.exceptions.CliArrError*

exception pycliarr.api.exceptions.**CliServerError**(*message: str, status_code: int, response: str*)
Bases: *pycliarr.api.exceptions.CliArrError*

exception pycliarr.api.exceptions.**RadarrCliError**
Bases: *pycliarr.api.exceptions.CliArrError*

exception pycliarr.api.exceptions.**SonarrCliError**
Bases: *pycliarr.api.exceptions.CliArrError*

pycliarr.api.radarr module

class pycliarr.api.radarr.**RadarrCli**(**args*, *default_root_folder_id: int = 0*, ***kwargs*)
Bases: *pycliarr.api.base_media.BaseCliMediaApi*

Radar api client.

Radarr API reference: https://github.com/Radarr/Radarr/wiki/API https://pub.dev/packages/radarr

Note: Not all commands are implemented. Some commands available are implemented in BaseCliMediaApi: * get_calendar * get_command * get_quality_profiles * rename_files * get_disk_space * get_system_status * get_queue * delete_queue * get_history * get_logs * get_wanted * get_blocklist * delete_blocklist

add_movie(*quality: int*, *tmdb_id: Optional[int] = None*, *imdb_id: Optional[str] = None*, *movie_info: Optional[pycliarr.api.radarr.RadarrMovieItem] = None*, *monitored: bool = True*, *search: bool = True*, *path: Optional[str] = None*, *root_id: int = 0*) → Union[Dict[str, Any], List[Dict[str, Any]]]

addMovie adds a new movie to collection.

The movie description movie_info must be specified. If the IMDB or TMDB id is provided instead, it will be used to fetch the required movie description from TMDB.

Parameters

- **quality** – Quality profile to use, as retrieved by get_quality_profiles()
- **imdbp_id** (*Optional[int]*) – IMDB id of the movie to add
- **tmdb_id** (*Optional[int]*) – TMDB id of the movie to add
- **movie_info** (*Optional[RadarrMovieItem]*) – Description of the movie to add
- **monitored** (*bool*) – Whether to monitor the movie. Default is True
- **search** (*bool*) – Whether to search for the movie once added. Default is True
- **path** (*Optional[str]*) – Specify the path awhere the movie should be stored. Default is root[0]/<movie name> (<movie year>).
- **root_id** (*Optional[int]*) – Specify the root folder to use. Ignored if a path is specified. Default is root[0].

Returns json response

api_url_exclusions = '/api/v3/exclusions'

api_url_item = '/api/v3/movie'

api_url_itemlookup = '/api/v3/movie/lookup'

api_url_wanted_missing = '/api/wanted/missing'

build_movie_path(*movie_info: pycliarr.api.radarr.RadarrMovieItem*, *root_folder_id: int = 0*) → pathlib.Path

Build a movie folder path using the root folder specified. :param serie_info: :type serie_info: SonarrSerieItem :param root_folder_id: Id of the root folder (can be retrieved with get_root_folder()) :type root_folder_id: int :param If the id is not found or not specified, the first root folder in the list is used.:

Returns: Full path of the serie in the format <root path>/<movie name> (<movie year>)

create_exclusion(*title: str*, *tmdb_id: int*, *year: int*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Create the specified exclusions

Parameters **item_id** (*int*) – id of the exclusions to create

Returns json response

delete_movie(*movie_id: int*, *delete_files: bool = True*, *add_exclusion: bool = False*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Delete the movie with the given ID

Parameters

- **movie_id** (*int*) – Movie to delete
- **delete_files** (*bool*) – Optional. Also delete files. Default is True
- **add_exclusion** – Optionally exclude the movie from further imdb/tmdb auto add

Returns json response

edit_movie(*movie_info: pycliarr.api.radarr.RadarrMovieItem*, *move_files: bool = False*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Edit a movie from the collection.

The movie description movie_info must be specified, usually by getting the information from get_movie()

Parameters

- **movie_info** (*Optional[RadarrMovieItem]*) – Description of the movie to edit
- **move_files** (*bool*) – Whether to move files after edition. Default is False

Returns json response

get_movie(*movie_id: Optional[int] = None*) → Union[pycliarr.api.radarr.RadarrMovieItem, List[pycliarr.api.radarr.RadarrMovieItem]]

Get specified movie, or all if no id provided from server collection.

Parameters **movie_id** (*Optional[int]*) –

Returns `RadarrMovieItem` if a movie id is specified, or a list of `RadarrMovieItem`

get_queue(*page: int = 1*, *sort_key: str = 'progress'*, *page_size: int = 20*, *sort_dir: str = 'ascending'*, *include_unknown: bool = True*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Get queue info (downloading/completed, ok/warning) as json

Parameters

- **page** (*int) - 1-indexed (1 default*) –
- **sort_key** (*string*) –
- **page_size** (*int*) – 10
- **sort_dir** (*string*) – asc
- **options** (*Dict[str, Any]={}*) – Optional additional options

lookup_movie(*term: Optional[str] = None*, *imdb_id: Optional[str] = None*, *tmdb_id: Optional[int] = None*) → Union[pycliarr.api.radarr.RadarrMovieItem, List[pycliarr.api.radarr.RadarrMovieItem], None]

Search for a movie based on keyword, or imbd/tmdb id.

If no imdb id is provided, tvdb id will be used. If neither of them is provided, the keyword will be used. One of `term`, `imdb_id`, or `tmdb_id` must be specified.

Parameters

- **term** (*Optional[str]*) – Keywords to seach for
- **imdb_id** (*Optional[str]*) – IMDB movie id
- **tmdb_id** (*Optional[int]*) – TMDB movie id

Returns json response

missing_movies_search() → Union[Dict[str, Any], List[Dict[str, Any]]]

Search for missing movies. :returns: json response

refresh_movie(*movie_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Refresh movie information and rescan disk.

Parameters **movie_id** (*Optional[int]*) – Movie to refresh, if not set all movies will be refreshed and scanned

Returns json response

rescan_movie(*movie_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Scan disk for any downloaded movie for all or specified movie.

Parameters **movie_id** (*Optional[int]*) – Movie to refresh, if not set all movies will be refreshed and scanned

Returns json response

class pycliarr.api.radarr.**RadarrMovieItem**(***kwargs*)
Bases: *pycliarr.api.base_api.BaseCliApiItem*

Class for handling movie info.

pycliarr.api.sonarr module

class pycliarr.api.sonarr.**SonarrCli**(**args*, *default_root_folder_id: int = 0*, ***kwargs*)
Bases: *pycliarr.api.base_media.BaseCliMediaApi*

Sonarr api client.

API reference: https://github.com/Sonarr/Sonarr/wiki/API https://pub.dev/packages/sonarr

---

Note: Not all commands are implemented. Some commands available are implemented in BaseCliMediaApi: * get_calendar * get_command * get_quality_profiles * rename_files * get_disk_space * get_system_status * get_queue * delete_queue

---

Todo:

- get_wanted
- get_logs
- get_backup
- get_episode_files
- delete_episode_files
- search_selected

---

add_serie(*quality: int*, *tvdb_id: Optional[int] = None*, *serie_info: Optional[pycliarr.api.sonarr.SonarrSerieItem] = None*, *monitored_seasons: List[int] = []*, *monitored: bool = True*, *search: bool = True*, *season_folder: bool = True*, *path: Optional[str] = None*, *root_id: int = 0*, *language: int = 1*) → Union[Dict[str, Any], List[Dict[str, Any]]]
addMovie adds a new serie to collection.

The serie description serie_info must be specified. If the IMDB or TMDB id is provided instead, it will be used to fetch the required serie description from TMDB.

Parameters

- **quality** – Quality profile to use, as retrieved by get_quality_profiles()
- **tvdb_id** (*Optional[int]*) – TVDB id of the serie to add

- **serie_info** (*Optional[RadarrserieItem]*) – Description of the serie to add
- **monitored_seasons** – Optional list of seasons numbers to monitor. Latest season only by default.
- **monitored** (*bool*) – Whether to monitor the serie. Default is True
- **search** (*bool*) – Whether to search for the serie once added. Default is True
- **season_folder** (*bool*) – If True (default), create a folder for each season.
- **path** (*Optional[str]*) – Specify the path awhere the movie should be stored. Default is root/<serie name>.
- **root_id** (*Optional[int]*) – Specify the root folder to use. Ignored if a path is specified. Default is root[0].
- **language** (*int*) – Specify the language to use. Default is the first enabled (1)

Returns json response

Note: To further customize the parameters of the serie to add, manually look it up .. rubric:: Example

info = sonarr.lookup_serie(tvdb_id=tvdb_id) info[“seasons”] = {“seasonNumber”: 1, “monitored”: False} sonarr.add_serie(quality: 1, serie_info: info)

api_url_episode = '/api/v3/episode'

api_url_episodefile = '/api/v3/episodefile'

api_url_item = '/api/v3/series'

api_url_itemlookup = '/api/v3/series/lookup'

api_url_wanted_missing = '/api/wanted/missing'

build_serie_path(*serie_info: pycliarr.api.sonarr.SonarrSerieItem*, *root_folder_id: int = 0*) → pathlib.Path

Build a serie folder path using the root folder specified. :param serie_info: :type serie_info: SonarrSerieItem :param root_folder_id: Id of the root folder (can be retrieved with get_root_folder()). :type root_folder_id: int :param If the id is not found or not specified, the first root folder in the list is used.:

Returns: Full path of the serie in the format <root path>/<serie name>

create_exclusion(*title: str*, *tvdb_id: int*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Create the specified exclusions

Parameters **item_id** (*int*) – id of the exclusions to create

Returns json response

delete_episode_file(*episode_id: int*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Delete the given episode file

Parameters **episode_id** (*int*) – ID of the episode to delete

Returns json response

delete_serie(*serie_id: int*, *delete_files: bool = True*, *add_exclusion: bool = False*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Delete the serie with the given ID

Parameters

- **serie_id** (*int*) – Serie to delete
- **delete_files** (*bool*) – Optional. Also delete files. Default is True

- **add_exclusion** – Optionally exclude the serie from further tvdb auto add

Returns json response

edit_serie(*serie_info: pycliarr.api.sonarr.SonarrSerieItem*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Edit a serie from the collection.

The serie description movie_info must be specified, usually by getting the information from get_serie()

Parameters **serie_info** (*Optional[RadarrMovieItem]*) – Description of the movie to edit

Returns json response

get_episode(*serie_id: Optional[int] = None*, *episode_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]], List[Union[Dict[str, Any], List[Dict[str, Any]]]]]
Returns specified episode or all for the given serie

Parameters

- **serie_id** (*int*) – ID of the serie to get all episodes from
- **episode_id** (*int*) – ID of a specific episode to get

Returns json response

get_episode_file(*serie_id: Optional[int] = None*, *episode_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]], List[Union[Dict[str, Any], List[Dict[str, Any]]]]]
Returns specified episode file or all for the given serie

Parameters

- **serie_id** (*int*) – ID of the serie to get all episodes files from
- **episode_id** (*int*) – ID of a specific episode file to get

Returns json response

get_queue(*page: int = 1*, *sort_key: str = 'progress'*, *page_size: int = 20*, *sort_dir: str = 'ascending'*, *include_unknown: bool = True*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Get queue info (downloading/completed, ok/warning) as json

Parameters

- **page** (*int) - 1-indexed (1 default*) –
- **sort_key** (*string*) –
- **page_size** (*int*) – 10
- **sort_dir** (*string*) – asc
- **options** (*Dict[str, Any]={}*) – Optional additional options

get_serie(*serie_id: Optional[int] = None*) → Union[pycliarr.api.sonarr.SonarrSerieItem, List[pycliarr.api.sonarr.SonarrSerieItem]]
Get specified serie, or all if no id provided from server collection.

Parameters **serie_id** (*Optional[int]*) –

Returns SonarrSerieItem if a serie id is specified, or a list of SonarrSerieItem

lookup_serie(*term: Optional[str] = None*, *tvdb_id: Optional[int] = None*) → Union[pycliarr.api.sonarr.SonarrSerieItem, List[pycliarr.api.sonarr.SonarrSerieItem], None]
Search for a serie based on keyword, or tvdb id.

If tvdb id is provided, it will be used. If not, the keywords will be used. One of `term`, or `tvdb_id` must be specified.

Parameters

- **term** (*Optional[str]*) – Keywords to seach for
- **tvdb_id** (*Optional[str]*) – TVDB serie id

Returns json response

missing_episodes_search() → Union[Dict[str, Any], List[Dict[str, Any]]]
Search for missing episodes. :returns: json response

refresh_serie(*serie_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Refresh serie information and rescan disk.

Parameters **serie_id** (*Optional[int]*) – serie to refresh, if not set all series will be refreshed and scanned

Returns json response

rescan_serie(*serie_id: Optional[int] = None*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Scan disk for any downloaded serie for all or specified serie.

Parameters **serie_id** (*Optional[int]*) – serie to refresh, if not set all series will be refreshed and scanned

Returns json response

class pycliarr.api.sonarr.**SonarrSerieItem**(**kwargs)
Bases: *pycliarr.api.base_api.BaseCliApiItem*

Class for handling serie info.

Module contents

pycliarr.cli package

Submodules

pycliarr.cli.cli module

pycliarr.cli.cli.**main**() → None
Main entry point.

pycliarr.cli.cli_cmd module

class pycliarr.cli.cli_cmd.**CliAddMovieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Add a movie from the imdb/tmdb id, or look for keywords'

name = 'add'

run(*cli: pycliarr.api.radarr.RadarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliAddSerieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Add a serie from the tvdb id, or look for keywords'

name = 'add'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliApiCommand**(*name: str*, *cli_class: Any*, *commands: List[pycliarr.cli.cli_cmd.CliCommand]*)
Bases: object

Definition of an API client.

Allows instantiating the relevant communication client, and execute a subcommmand from its name.

add_commands_args(*cmd_subparser: argparse._SubParsersAction*) → None

run_command(*cmd_name: str*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliCommand**
Bases: object

Base command, all command should extend this class.

configure_args(*cmdlist_parser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = ''

name = ''

run(*cli: Any*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliCreateRadarrExclusionCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Create the specified exclusion'

name = 'create-exclusion'

run(*cli: pycliarr.api.radarr.RadarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliCreateSonarrExclusionCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Create the specified exclusion'

name = 'create-exclusion'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliCreateTagCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Create the specified tag'

name = 'create-tag'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliDeleteBlocklistCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get list of quality profiles'

name = 'delete-blocklist'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliDeleteEpisodeFileCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get info on a of serie'

name = 'delete-episode-file'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliDeleteExclusionCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Delete the specified exclusion'

name = 'delete-exclusion'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliDeleteMovieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Delete a movie'

name = 'delete'

run(*cli: pycliarr.api.radarr.RadarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliDeleteNotificationCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Delete the specified notification or all'

name = 'delete-notification'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliDeleteQueueCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get list of quality profiles'

name = 'delete-queue'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliDeleteSerieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Delete a serie'

name = 'delete'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliDeleteTagCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Delete the specified tag'

name = 'delete-tag'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliEditMovieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Push an updated item to the movie library'

name = 'edit'

run(*cli: pycliarr.api.radarr.RadarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliEditTagCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Edit the specified tag'

name = 'edit-tag'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliEpisodeCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get info on an episode'

name = 'get-episode'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetBlocklistCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get blocklisted items'

name = 'blocklist'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetCalendarCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get events from calendar'

name = 'calendar'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetDiskSpaceCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

description = 'Get disk space'

name = 'disk-space'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetEpisodeFileCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get info on an episode file'

name = 'get-episode-file'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetExclusionCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get exclusion(s)'

name = 'exclusion'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetMovieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get info on a of movie'

name = 'get'

run(*cli: pycliarr.api.radarr.RadarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetNotificationCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get notification(s)'

name = 'notification'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetProfilesCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

description = 'Get list of quality profiles'

name = 'profiles'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetQueueCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get current downloading queue'

name = 'queue'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetRefreshMovieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Refresh movies'

name = 'refresh'

run(*cli: pycliarr.api.radarr.RadarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetRefreshSerieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Refresh series'

name = 'refresh'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetRescanMovieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Rescan movies'

name = 'rescan'

run(*cli: pycliarr.api.radarr.RadarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetRescanSerieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Rescan series'

name = 'rescan'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetSerieCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get info on a of serie'

name = 'get'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetTagCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get tag(s)'

name = 'tag'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetTagDetailCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get tag(s) details'

name = 'tag-detail'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliGetTagItemsCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'List items with specifed tag'

name = 'tag-items'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliPutNotificationCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Create the specified notification'

name = 'put-notification'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliRootFoldersCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get root folder list'

name = 'root-folders'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliSearchMissingEpisodes**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

description = 'Search missing episods'

name = 'search-missing'

run(*cli: pycliarr.api.sonarr.SonarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliSearchMissingMovies**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

description = 'Search missing movies'

name = 'search-missing'

run(*cli: pycliarr.api.radarr.RadarrCli*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliStatusCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'Get status of 1 or all currently running commands'

name = 'status'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliSystemStatusCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

description = 'Get system status'

name = 'system-status'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

class pycliarr.cli.cli_cmd.**CliWantedCommand**
Bases: *pycliarr.cli.cli_cmd.CliCommand*

configure_args(*cmd_subparser: argparse._SubParsersAction*) → argparse.ArgumentParser

description = 'List wanted/missing'

name = 'wanted'

run(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *args: argparse.Namespace*) → None

pycliarr.cli.cli_cmd.**print_root_folder**(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *raw=<class 'bool'>*) → None

pycliarr.cli.cli_cmd.**root_folder_id_from_arg**(*cli: pycliarr.api.base_media.BaseCliMediaApi*, *root_arg: str*) → int

pycliarr.cli.cli_cmd.**select_item**(*terms: str, choices: List[Union[pycliarr.api.radarr.RadarrMovieItem, pycliarr.api.sonarr.SonarrSerieItem]]*) → Union[pycliarr.api.radarr.RadarrMovieItem, pycliarr.api.sonarr.SonarrSerieItem]

pycliarr.cli.cli_cmd.**select_language_profile**(*cli: pycliarr.api.base_media.BaseCliMediaApi*) → int

pycliarr.cli.cli_cmd.**select_profile**(*cli: pycliarr.api.base_media.BaseCliMediaApi*) → int

pycliarr.cli.cli_cmd.**select_root_folder**(*cli: pycliarr.api.base_media.BaseCliMediaApi*) → int

pycliarr.cli.utils module

pycliarr.cli.utils.**setup_logging**(*level: int = 20*, *filename: Optional[str] = None*) → None
Configure standard logging.

pycliarr.cli.utils.**size_to_str**(*size: Optional[float]*) → str

Module contents

7.1.2 Module contents

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

A

B

C

D

P

R

S

T