anime-list-apis

Release 0.3.0

Feb 02, 2019

CONTENTS

1 anime_list_apis **3**
1.1 anime_list_apis package 3

2 Indices and tables **29**

Python Module Index **31**

Index **33**

Contents:

CHAPTER

ONE

ANIME_LIST_APIS

1.1 anime_list_apis package

1.1.1 Subpackages

anime_list_apis.api package

Submodules

anime_list_apis.api.AnilistApi module

class anime_list_apis.api.AnilistApi.**AnilistApi**(*cache: anime_list_apis.cache.Cache.Cache = None*, *rate_limit_pause: float = 0.5*)

Bases: *anime_list_apis.api.ApiInterface.ApiInterface*

Implements a wrapper around the anilist.co API

__init__(*cache: anime_list_apis.cache.Cache.Cache = None*, *rate_limit_pause: float = 0.5*)

Initializes the Anilist Api interface. Intializes cache or uses the one provided. :param cache: The cache to use. If left as None, will use default cache :param rate_limit_pause: A duration in seconds that the API Interface

will pause after a network operation to prevent being rate limited

get_anilist_id_from_mal_id(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *mal_id: int*) → Optional[int]

Retrieves an anilist ID from a myanimelist ID :param media_type: The media type of the myanimelist ID :param mal_id: The myanimelist ID :return: The anilist ID. May be None if myanimelist ID has no

equivalent on anilist

anime_list_apis.api.ApiInterface module

class anime_list_apis.api.ApiInterface.**ApiInterface**(*id_type: anime_list_apis.models.attributes.Id.IdType*, *cache: anime_list_apis.cache.Cache.Cache = None*, *rate_limit_pause: float = 0.0*)

Bases: `object`

Defines methods that every API connector should implement

__init__(*id_type: anime_list_apis.models.attributes.Id.IdType*, *cache: anime_list_apis.cache.Cache.Cache = None*, *rate_limit_pause: float = 0.0*)
Initializes the Api interface. Intializes cache or uses the one provided. :param id_type: TheID type of the API Interface :param cache: The cache to use. If left as None, will use default cache :param rate_limit_pause: A duration in seconds that the API Interface

will pause after a network operation to prevent being rate limited

get_anime_data(*_id: int*, *fresh: bool = False*) → Optional[anime_list_apis.models.MediaData.AnimeData]
Shortcut for get_data that retrieves only Anime media :param _id: The ID to fetch. May be int or Id object :param fresh: Fetches a fresh, i.e. non-cached version :return: The retrieved AnimeData object or None if not a valid ID

get_anime_list(*username: str*) → List[anime_list_apis.models.MediaListEntry.AnimeListEntry]
Retrieves a user's complete anime list :param username: The user's username :return: The user's list of AnimeListEntry objects

get_anime_list_entry(*_id: int*, *username: str*, *fresh: bool = False*) → Optional[anime_list_apis.models.MediaListEntry.AnimeListEntry]
Retrieves a user's list entry for a single entry :param _id: The ID to fetch. May be int or an Id object :param username: The user for which to fetch the entry :param fresh: Fetches a fresh, i.e. non-cached version :return: The Anime List Entry or None if there is not such entry

get_anime_user_data(*_id: int*, *username: str*, *fresh: bool = False*) → Optional[anime_list_apis.models.MediaUserData.AnimeUserData]
Retrieves a user's user data for a single entry :param _id: The ID to fetch. May be int or an Id object :param username: The user for which to fetch the user data :param fresh: Fetches a fresh, i.e. non-cached version :return: The AnimeUserData or None if there is not such entry

get_anime_user_data_list(*username: str*) → List[anime_list_apis.models.MediaUserData.AnimeUserData]
Retrieves a user's complete anime list of user data objects :param username: The user's username :return: The user's list of AnimeUserData objects

get_data(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *fresh: bool = False*) → Optional[anime_list_apis.models.MediaData.MediaData]
Retrieves a single data object using the API Tries to get the cached value first, then checks Anilist :param media_type: The media type to retrieve :param _id: The ID to retrieve. May be either an int or an Id object :param fresh: Fetches a fresh, i.e. non-cached version :return: The Media Data or None if no valid data was found

get_list(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *username: str*) → List[anime_list_apis.models.MediaListEntry.MediaListEntry]
Retrieves a user's entire list Stores all entries in the cache upon completion :param media_type: The media type to fetch :param username: The username for which to fetch the list :return: The list of List entries

get_list_entry(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: str*, *fresh: bool = False*) → Optional[anime_list_apis.models.MediaListEntry.MediaListEntry]
Retrieves a user list entry. First checks for cached entries, otherwise fetches from anilist :param media_type: The media type to fetch :param _id: The ID to retrieve. May be and int or an Id object :param username: The user for which to fetch the entry :param fresh: Fetches a fresh, i.e. non-cached version :return: The entry for the user or

None if the user doesn't have such an entry

get_manga_data(*_id: int*, *fresh: bool = False*) → Optional[anime_list_apis.models.MediaData.MangaData]
Shortcut for get_data that retrieves only Manga media :param _id: The ID to fetch. May be int or Id object :param fresh: Fetches a fresh, i.e. non-cached version :return: The retrieved MangaData object or None if not a valid ID

get_manga_list(*username: str*) → List[anime_list_apis.models.MediaListEntry.MangaListEntry]
Retrieves a user's complete manga list :param username: The user's username :return: The user's list of MangaListEntry objects

get_manga_list_entry(*_id: int*, *username: str*, *fresh: bool = False*) → Optional[anime_list_apis.models.MediaListEntry.MangaListEntry]
Retrieves a user's list entry for a single entry :param _id: The ID to fetch. May be int or an Id object :param username: The user for which to fetch the entry :param fresh: Fetches a fresh, i.e. non-cached version :return: The Manga List Entry or None if there is not such entry

get_manga_user_data(*_id: int*, *username: str*, *fresh: bool = False*) → Optional[anime_list_apis.models.MediaUserData.MangaUserData]
Retrieves a user's user data for a single entry :param _id: The ID to fetch. May be int or an Id object :param username: The user for which to fetch the user data :param fresh: Fetches a fresh, i.e. non-cached version :return: The MangaUserData or None if there is not such entry

get_manga_user_data_list(*username: str*) → List[anime_list_apis.models.MediaUserData.AnimeUserData]
Retrieves a user's complete manga list of user data objects :param username: The user's username :return: The user's list of MangaUserData objects

get_related_data(*datas: List[anime_list_apis.models.MediaData.MediaData]*, *fresh: bool = False*) → List[anime_list_apis.models.MediaData.MediaData]
Retrieves related data for either a list of MediaData objects or a single MediaData object :param datas: A list of MediaData objects (or a single one) :param fresh: Indicates if the most up-to-date results should be used :return: A list of related media data

get_user_data(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: str*, *fresh: bool = True*) → Optional[anime_list_apis.models.MediaUserData.MediaUserData]
Retrieves the user data of a single entry for a user :param media_type: The type of media to fetch :param _id: The ID to fetch :param username: The username for which to fetch :param fresh: Fetches a fresh, i.e. non-cached version :return: The MediaUserData object or None if not found

get_user_data_list(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *username: str*) → List[anime_list_apis.models.MediaUserData.MediaUserData]
Retrieves a user's entire list with only the user data :param media_type: The media type to fetch the entries for :param username: The user for whom to fetch the entries for :return: The retrieves user data in a list

is_in_anime_list(*_id: int*, *username: str*, *fresh: bool = False*) → bool
Checks if an entry is in a user's list :param _id: The ID to check for :param username: The username for which to check :param fresh: Indicates if the most up-to-date results should be used :return: True if the id is in the list, False otherwise

is_in_list(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: str*, *fresh: bool = False*) → bool
Checks if an entry is in a user's list :param media_type: The media type to check for :param _id: The ID to check for :param username: The username for which to check :param fresh: Indicates if the most up-to-date results should be used :return: True if the id is in the list, False otherwise

is_in_manga_list(*_id: int*, *username: str*, *fresh: bool = False*) → bool
Checks if an entry is in a user's manga list :param _id: The ID to check for :param username: The username for which to check :param fresh: Indicates if the most up-to-date results should be used :return: True if the id is in the list, False otherwise

anime_list_apis.api.KitsuApi module

class anime_list_apis.api.KitsuApi.**KitsuApi**(*cache: anime_list_apis.cache.Cache.Cache = None*, *rate_limit_pause: float = 0.0*)
Bases: *anime_list_apis.api.ApiInterface.ApiInterface*

Implements a wrapper around the kitsu.io API

__init__(*cache: anime_list_apis.cache.Cache.Cache = None*, *rate_limit_pause: float = 0.0*)
Initializes the Kitsu Api interface. Intializes cache or uses the one provided. :param cache: The cache to use. If left as None, will use default cache :param rate_limit_pause: A duration in seconds that the API Interface

will pause after a network operation to prevent being rate limited

anime_list_apis.api.MyanimelistApi module

class anime_list_apis.api.MyanimelistApi.**MyanimelistApi**(*cache: anime_list_apis.cache.Cache.Cache = None*, *rate_limit_pause: float = 0.0*)
Bases: *anime_list_apis.api.ApiInterface.ApiInterface*

Implements a wrapper around the myanimelist.net API

__init__(*cache: anime_list_apis.cache.Cache.Cache = None*, *rate_limit_pause: float = 0.0*)
Initializes the Myanimelist Api interface. Intializes cache or uses the one provided. :param cache: The cache to use. If left as None, will use default cache :param rate_limit_pause: A duration in seconds that the API Interface

will pause after a network operation to prevent being rate limited

Module contents

anime_list_apis.cache package

Submodules

anime_list_apis.cache.Cache module

class anime_list_apis.cache.Cache.**Cache**(*cache_location: str = None*, *expiration: int = 6000*, *write_after: int = 20*)
Bases: `object`

Handles various caching functionality

__init__(*cache_location: str = None*, *expiration: int = 6000*, *write_after: int = 20*)
Initializes the Cache. If the Cache directory and file do not exist, they will be created here. :param cache_location: The location of the cache. Will default to a

hidden directory in the user's home directory

Parameters

- **expiration** – Defines how long objects should be valid. If set to a negative number, will be infinite
- **write_after** – Defines after how many cache changes the changes are automatically written to the cache file

add(*site_type: anime_list_apis.models.attributes.Id.IdType*, *data: anime_list_apis.models.CacheAble.CacheAble*, *ignore_for_write_count: bool = False*)
Adds a copy of an object to the cache. If the amount of changes exceeds the amount defined in write_after,

write to file afterwards :param site_type: The site for which to cache it :param data: The data to cache :param ignore_for_write_count: If set to True, will not increment the

change_count variable.

Returns None

add_primitive(*site_type: anime_list_apis.models.attributes.Id.IdType*, *key: str*, *value: Any*)
Adds a primitive data object to the cache :param site_type: The site for which to cache :param key: The key of the value :param value: the value to cache :return: None

static generate_id_tag(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: Optional[str] = None*) → str
Generates an ID tag for the cache :param media_type: The media type of the tag :param _id: The ID of the entry :param username: Optionally create a tag for a specific username :return: The generated ID tag

get(*model_type: anime_list_apis.models.CacheAble.CacheModelType*, *site_type: anime_list_apis.models.attributes.Id.IdType*, *media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: Optional[str] = None*) → Optional[anime_list_apis.models.CacheAble.CacheAble]
Retrieves a cached object. If the object has expired, remove it from the cache :param model_type: The cached data type :param site_type: The site type for which the object was cached :param media_type: The media type of the object to get :param _id: The ID to search for :param username: Optional-The username associated with the data object :return: A copy of the cached object, or None if it wasn't found

get_media_data(*site_type: anime_list_apis.models.attributes.Id.IdType*, *media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*) → Optional[anime_list_apis.models.MediaData.MediaData]
Retrieves a media data entry from the cache :param site_type: The site for which to fetch an entry :param media_type: The type of the media :param _id: The ID to fetch :return: The entry data or None if no corresponding entry exists

get_media_list_entry(*site_type: anime_list_apis.models.attributes.Id.IdType*, *media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: str*) → Optional[anime_list_apis.models.MediaListEntry.MediaListEntry]
Retrieves a media list entry from the cache :param site_type: The site for which to fetch the cached object :param media_type: The media type of the entry :param _id: The ID of the entry :param username: The username of the entry :return: The entry or None if not found

get_media_user_data(*site_type: anime_list_apis.models.attributes.Id.IdType*, *media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: str*) → Optional[anime_list_apis.models.MediaUserData.MediaUserData]
Retrieves a media user entry from the cache :param site_type: The site type for which the object was cached :param media_type: The media type of the cached object :param _id: The ID of the corresponding media data :param username: The username of the entry :return: The entry or None if not found

get_primitive(*site_type: anime_list_apis.models.attributes.Id.IdType*, *key: str*) → Any
Retrieves a primitive data object from the cache :param site_type: The site for which to get the cached data :param key: The key to retrieve :return: The cached primitive data object or None if no entry in cache

invalidate(*model_type: anime_list_apis.models.CacheAble.CacheModelType*, *site_type: anime_list_apis.models.attributes.Id.IdType*, *media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: Optional[str] = None*)
Invalidates a cache entry :param model_type: The model type of the entry to invalidate :param site_type: The site type of the entry to invalidate :param media_type: The media type of the entry :param _id: The ID of the entry :param username: The username for which to invalidate the entry :return: None

invalidate_media_data(*site_type: anime_list_apis.models.attributes.Id.IdType*, *media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*)
Invalidates a media entry :param site_type: The site type of the entry :param media_type: The media type of the entry :param _id: The ID of the entry :return: None

invalidate_media_list_entry(*site_type: anime_list_apis.models.attributes.Id.IdType*, *media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: str*)
Invalidates a list entry :param site_type: The site type of the entry :param media_type: The media type of the entry :param _id: The ID of the entry :param username: The user for which to invalidate the entry :return: None

invalidate_media_user_data(*site_type: anime_list_apis.models.attributes.Id.IdType*, *media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *_id: int*, *username: str*)
Invalidates a user data entry :param site_type: The site type of the entry :param media_type: The media type of the entry :param _id: The ID of the entry :param username: The user for which to invalidate the entry :return: None

load()
Loads the content of the cache file into memory :return: None

model_map = {<CacheModelType.MEDIA_USER_DATA: 2>: <class 'anime_list_apis.models.Media
Maps model types to their respective classes

write()
Writes the content of the cache to the cache file :return: None

Module contents

anime_list_apis.models package

Subpackages

anime_list_apis.models.attributes package

Submodules

anime_list_apis.models.attributes.ConsumingStatus module

class anime_list_apis.models.attributes.ConsumingStatus.**ConsumingStatus**
Bases: enum.Enum

Enum that specifies the watching/reading state of a user's list entry

COMPLETED = 3

CURRENT = 1

DROPPED = 4

PAUSED = 5

PLANNING = 2

REPEATING = 6

anime_list_apis.models.attributes.Date module

class anime_list_apis.models.attributes.Date.**Date**(*year: int*, *month: int*, *day: int*)
Bases: *anime_list_apis.models.Serializable.Serializable*

Class that models a date consisting of a year, a month and a day

__init__(*year: int*, *month: int*, *day: int*)
Initializes the date object. Each parameter must be filled out. :param year: The year :param month: The month :param day: The day :raises TypeError: If any of the parameters is not an integer :raises ValueError: If any of the parameters takes on an invalid value

anime_list_apis.models.attributes.Id module

class anime_list_apis.models.attributes.Id.**Id**(*ids: Dict[anime_list_apis.models.attributes.Id.IdType, int]*)
Bases: *anime_list_apis.models.Serializable.Serializable*

Class that models an ID. Has the capability to store different ID types

__init__(*ids: Dict[anime_list_apis.models.attributes.Id.IdType, int]*)
Initializes the ID. The IDs are set using a dictionary mapping IDs to the different ID types. At least one entry is required. Missing IDs will be replaced with None :param ids: The IDs mapped to an IdType :raises TypeError: If an invalid parameter type was provided :raises ValueError: In case no valid ID was provided

get(*id_type: anime_list_apis.models.attributes.Id.IdType*) → Optional[int]
Retrieves an ID for a given ID type :param id_type: The ID type to get :return: The ID. If it does not exist, return None

set(*_id: int*, *id_type: anime_list_apis.models.attributes.Id.IdType*)
Sets an ID for a given ID type :param _id: The ID to set :param id_type: The type of ID for which to set the ID :return: None :raises TypeError: If the provided ID is not an integer

class anime_list_apis.models.attributes.Id.**IdType**
Bases: enum.Enum

Enumeration of different ID types

ANILIST = 2

KITSU = 3

MYANIMELIST = 1

anime_list_apis.models.attributes.MediaFormat module

class anime_list_apis.models.attributes.MediaFormat.**MediaFormat**
Bases: enum.Enum

Enumeration that models possible media formats

LIGHT_NOVEL = 'NOVEL'

MANGA = 'MANGA'

MOVIE = 'MOVIE'

MUSIC = 'MUSIC'

ONA = 'ONA'

ONE_SHOT = 'ONE_SHOT'

OVA = 'OVA'

SPECIAL = 'SPECIAL'

TV = 'TV'

TV_SHORT = 'TV_SHORT'

anime_list_apis.models.attributes.MediaType module

class anime_list_apis.models.attributes.MediaType.**MediaType**
Bases: enum.Enum

Enumeration that models the different media types

ANIME = 'anime'

MANGA = 'manga'

anime_list_apis.models.attributes.Relation module

class anime_list_apis.models.attributes.Relation.**Relation**(*source: anime_list_apis.models.attributes.Id.Id*, *source_type: anime_list_apis.models.attributes.MediaType.Me*, *dest: anime_list_apis.models.attributes.Id.Id*, *dest_type: anime_list_apis.models.attributes.MediaType.Me*, *relation_type: anime_list_apis.models.attributes.Relation.Relat*)
Bases: *anime_list_apis.models.Serializable.Serializable*

Class that models a relation edge between two anime entries

__init__(*source: anime_list_apis.models.attributes.Id.Id*, *source_type: anime_list_apis.models.attributes.MediaType.MediaType*, *dest: anime_list_apis.models.attributes.Id.Id*, *dest_type: anime_list_apis.models.attributes.MediaType.MediaType*, *relation_type: anime_list_apis.models.attributes.Relation.RelationType*)
Initializes the Relation object. :param source: The source node in the relation edge :param source_type: The media type of the source node :param dest: The destination node in the relation edge :param dest_type: The destination node's media type :param relation_type: The type of the relation :raises TypeError: If invalid ID types are provided

or other types mismatch

Raises **ValueError** – If both IDs are the same, i.e. an invalid relation

is_important() → bool
Checks if this relation is "important", meaning if it's a direct connection (Sequel, Prequel, Parent, Side Story or a summary) or not :return: True if the relation is important, False otherwise

class anime_list_apis.models.attributes.Relation.**RelationType**
Bases: enum.Enum

An enumeration modelling the different kinds of relations between entries.

ADAPTATION = 203

ALTERNATIVE = 204

CHARACTER = 200

OTHER = 202

PARENT = 102

PREQUEL = 100

SEQUEL = 101

SIDE_STORY = 103

SPIN_OFF = 201

SUMMARY = 104

anime_list_apis.models.attributes.ReleasingStatus module

class anime_list_apis.models.attributes.ReleasingStatus.**ReleasingStatus**
Bases: enum.Enum

Enumeration that models the airing state of a media entry

CANCELLED = 4

FINISHED = 1

NOT_RELEASED = 3

RELEASING = 2

anime_list_apis.models.attributes.Score module

class anime_list_apis.models.attributes.Score.**Score**(*score: int*, *score_type: anime_list_apis.models.attributes.Score.ScoreType*)
Bases: *anime_list_apis.models.Serializable.Serializable*

Class that models a score. Allows for different score types.

__init__(*score: int*, *score_type: anime_list_apis.models.attributes.Score.ScoreType*)
Initializes the Score object. The score must be a positive number that less or equal than the maximum score of the score type. :param score: The points of the score :param score_type: The Type of score :raises TypeError: In case any of the parameter values

is of the wrong type

Raises **ValueError** – If the score is outside the valid range

convert(*score_type: anime_list_apis.models.attributes.Score.ScoreType*)
Converts the internal score representation to another score type :param score_type: The score type to which to convert to :return: None

get(*score_type: anime_list_apis.models.attributes.Score.ScoreType = None*) → int
Retrieves the score converted to the selected score type. If no score type was supplied, returns the default score type's score :param score_type: The score type in which to retrieve the score.

If left as None, will result in the default score type to be used

Returns The score in the provided score type

class anime_list_apis.models.attributes.Score.**ScoreType**
Bases: enum.Enum

Enumeration modelling the different score types The value of the enum specifies the maximum value a score of that type may be. For example, TEN_POINT scores may range from 0 to 10

FIVE_POINT = 5

PERCENTAGE = 100

TEN_POINT = 10

TEN_POINT_DECIMAL = 20

THREE_POINT = 3

anime_list_apis.models.attributes.Title module

class anime_list_apis.models.attributes.Title.**Title**(*titles: Dict[anime_list_apis.models.attributes.Title.TitleType, str], default: anime_list_apis.models.attributes.Title.TitleType = <TitleType.ROMAJI: 1>*)
Bases: *anime_list_apis.models.Serializable.Serializable*

Models a title of an entry

__init__(*titles: Dict[anime_list_apis.models.attributes.Title.TitleType, str], default: anime_list_apis.models.attributes.Title.TitleType = <TitleType.ROMAJI: 1>*)
Initializes the Title object. In case no valid titles are supplied, a ValueError is raised :param titles: The titles to include :param default: The default title to show. Defaults to ROMAJI :raises TypeError: If an invalid parameter type was provided :raises ValueError: In case no valid titles were provided

change_default_title_type(*title_type: anime_list_apis.models.attributes.Title.TitleType*)
Sets the default title type :param title_type: The new default title type :return: None :raises ValueError: If there exists no title entry for the provided

title type

get(*title_type: anime_list_apis.models.attributes.Title.TitleType = None*) → Optional[str]
Retrieves the title in the provided title format :param title_type: The title type in which to retrieve the title.

Defaults to the current default title type

Returns The requested title string or None if no title string for the provided title exists

set(*title: str*, *title_type: anime_list_apis.models.attributes.Title.TitleType*)
Sets the title of a title type :param title: The title string to set :param title_type: The type of that title :return: None :raises TypeError: If the type of the title string is wrong

class anime_list_apis.models.attributes.Title.**TitleType**
Bases: enum.Enum

An enumeration modelling the different type of titles

ENGLISH = 2

JAPANESE = 3

ROMAJI = 1

Module contents

Submodules

anime_list_apis.models.CacheAble module

class anime_list_apis.models.CacheAble.**CacheAble**
Bases: *anime_list_apis.models.Serializable.Serializable*

Class that defines methods needed to be implemented by a cache-able object

generate_tag(*site_type: anime_list_apis.models.attributes.Id.IdType*) → str
Generates a tag/identifier for storing in the cache :param site_type: The site type for which to generate the tag :return: The generated tag

get_id() → anime_list_apis.models.attributes.Id.Id
Retrieves the cache entry's ID :return: The ID

get_media_type() → anime_list_apis.models.attributes.MediaType.MediaType
Retrieves the media type :return: The media type

get_model_type() → anime_list_apis.models.CacheAble.CacheModelType
Retrieves the cache model type :return: The model type

get_username() → Optional[str]
Retrieves the username, if applicable. Else None :return: The username or None if not applicable

class anime_list_apis.models.CacheAble.**CacheModelType**
Bases: enum.Enum

An enumeration that keeps track of different cache-able model types

DATA = 0

MEDIA_DATA = 1

MEDIA_LIST_ENTRY = 3

MEDIA_USER_DATA = 2

anime_list_apis.models.MediaData module

class anime_list_apis.models.MediaData.**AnimeData**(*_format: anime_list_apis.models.attributes.MediaFormat.MediaFormat, _id: anime_list_apis.models.attributes.Id.Id, title: anime_list_apis.models.attributes.Title.Title, relations: List[anime_list_apis.models.attributes.Relation.Relation], releasing_status: anime_list_apis.models.attributes.ReleasingStatus.ReleasingS, releasing_start: Optional[anime_list_apis.models.attributes.Date.Date], releasing_end: Optional[anime_list_apis.models.attributes.Date.Date], cover_url: Optional[str], episode_count: Optional[int], episode_duration: Optional[int]*)
Bases: *anime_list_apis.models.MediaData.MediaData*

Class that models anime data

__init__(*_format: anime_list_apis.models.attributes.MediaFormat.MediaFormat, _id: anime_list_apis.models.attributes.Id.Id, title: anime_list_apis.models.attributes.Title.Title, relations: List[anime_list_apis.models.attributes.Relation.Relation], releasing_status: anime_list_apis.models.attributes.ReleasingStatus.ReleasingStatus, releasing_start: Optional[anime_list_apis.models.attributes.Date.Date], releasing_end: Optional[anime_list_apis.models.attributes.Date.Date], cover_url: Optional[str], episode_count: Optional[int], episode_duration: Optional[int]*)

Initializes the Anime Data object and checks for type issues :param _format: The media format :param _id: The ID of the anime :param title: The title of the anime :param relations: The relations of the anime :param releasing_status: The airing status of the anime :param releasing_start: The day the anime started airing :param releasing_end: The day the last episode aired :param episode_count: The amount of episodes of the anime :param episode_duration: The duration of the episodes in minutes :param cover_url: An URL pointing to a cover image for the anime :raises TypeError: If any of the parameters has a wrong type

class anime_list_apis.models.MediaData.**MangaData**(*_format: anime_list_apis.models.attributes.MediaFormat.MediaFormat, _id: anime_list_apis.models.attributes.Id.Id, title: anime_list_apis.models.attributes.Title.Title, relations: List[anime_list_apis.models.attributes.Relation.Relation], releasing_status: anime_list_apis.models.attributes.ReleasingStatus.ReleasingS releasing_start: Optional[anime_list_apis.models.attributes.Date.Date], releasing_end: Optional[anime_list_apis.models.attributes.Date.Date], cover_url: Optional[str], chapter_count: Optional[int], volume_count: Optional[int]*)

Bases: *anime_list_apis.models.MediaData.MediaData*

Class that models manga data

__init__(*_format: anime_list_apis.models.attributes.MediaFormat.MediaFormat, _id: anime_list_apis.models.attributes.Id.Id, title: anime_list_apis.models.attributes.Title.Title, relations: List[anime_list_apis.models.attributes.Relation.Relation], releasing_status: anime_list_apis.models.attributes.ReleasingStatus.ReleasingStatus, releasing_start: Optional[anime_list_apis.models.attributes.Date.Date], releasing_end: Optional[anime_list_apis.models.attributes.Date.Date], cover_url: Optional[str], chapter_count: Optional[int], volume_count: Optional[int]*)

Initializes the Manga Data object and checks for type issues :param _format: The media format :param _id: The ID of the manga :param title: The title of the manga :param relations: The relations of the manga :param releasing_status: The releasing status of the manga :param releasing_start: The day the manga started releasing :param releasing_end: The day the last chapter/volume released :param chapter_count: The total amount of chapters of this manga :param volume_count: The total amount of volumes of this manga :param cover_url: An URL pointing to a cover image for the manga :raises TypeError: If any of the parameters has a wrong type

class anime_list_apis.models.MediaData.**MediaData**(*media_type: anime_list_apis.models.attributes.MediaType.MediaType, _format: anime_list_apis.models.attributes.MediaFormat.MediaFormat, _id: anime_list_apis.models.attributes.Id.Id, title: anime_list_apis.models.attributes.Title.Title, relations: List[anime_list_apis.models.attributes.Relation.Relation], releasing_status: anime_list_apis.models.attributes.ReleasingStatus.ReleasingStatus, releasing_start: Optional[anime_list_apis.models.attributes.Date.Date], releasing_end: Optional[anime_list_apis.models.attributes.Date.Date], cover_url: Optional[str]*)

Bases: *anime_list_apis.models.Serializable.MediaSerializable*, *anime_list_apis.models.CacheAble.CacheAble*

Class that models user-independent media data

__init__(*media_type: anime_list_apis.models.attributes.MediaType.MediaType, _format: anime_list_apis.models.attributes.MediaFormat.MediaFormat, _id: anime_list_apis.models.attributes.Id.Id, title: anime_list_apis.models.attributes.Title.Title, relations: List[anime_list_apis.models.attributes.Relation.Relation], releasing_status: anime_list_apis.models.attributes.ReleasingStatus.ReleasingStatus, releasing_start: Optional[anime_list_apis.models.attributes.Date.Date], releasing_end: Optional[anime_list_apis.models.attributes.Date.Date], cover_url: Optional[str]*)

Initializes the Media Data object and checks for type issues Some values may be None, for example if a media has not completed releasing yet. This constructor initializes all common values of all media types. :param media_type: The type of media :param _format: The media format :param _id: The ID of the media :param title: The title of the media :param relations: The relations of the media to other media :param releasing_status: The releasing status of the media :param releasing_start: The day the media started releasing :param releasing_end: The day the last content was released :param cover_url: An URL pointing to a cover image for the anime :raises TypeError: If any of the parameters has a wrong type

classmethod get_class_for_media_type(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*)

Maps a class to a media type :param media_type: The media type :return: The class mapped to that media type

get_id() → anime_list_apis.models.attributes.Id.Id

Retrieves the cache entry's ID :return: The ID

get_media_type() → anime_list_apis.models.attributes.MediaType.MediaType

Retrieves the media type :return: The media type

get_model_type() → anime_list_apis.models.CacheAble.CacheModelType

Retrieves the cache model type :return: The model type

get_username() → Optional[str]

Retrieves the username, if applicable. Else None :return: The username or None if not applicable

anime_list_apis.models.MediaListEntry module

class anime_list_apis.models.MediaListEntry.**AnimeListEntry**(*anime_data: anime_list_apis.models.MediaData.AnimeData*, *user_data: anime_list_apis.models.MediaUserData.AnimeUserData*)

Bases: *anime_list_apis.models.MediaListEntry.MediaListEntry*

Class that models a user's anime list entry

__init__(*anime_data: anime_list_apis.models.MediaData.AnimeData*, *user_data: anime_list_apis.models.MediaUserData.AnimeUserData*)

Initializes the anime list entry. :param anime_data: The anime data :param user_data: The user data :raises TypeError: If any of the parameters has a wrong type

get_media_data() → anime_list_apis.models.MediaData.AnimeData

Generates a new AnimeData object from the internal representation :return: The generated AnimeData object :raises TypeError: If any of the internal parameters has a wrong type

get_user_data() → anime_list_apis.models.MediaUserData.AnimeUserData

Generates a new AnimeUserData object from the internal representation :return: The generated AnimeUserData object :raises TypeError: If any of the internal parameters has a wrong type

class anime_list_apis.models.MediaListEntry.**MangaListEntry**(*manga_data: anime_list_apis.models.MediaData.MangaData*, *user_data: anime_list_apis.models.MediaUserData.MangaUserData*)

Bases: *anime_list_apis.models.MediaListEntry.MediaListEntry*

Class that models a user's manga list entry

__init__(*manga_data: anime_list_apis.models.MediaData.MangaData*, *user_data: anime_list_apis.models.MediaUserData.MangaUserData*)

Initializes the manga list entry. :param manga_data: The manga data :param user_data: The user data :raises TypeError: If any of the parameters has a wrong type

get_media_data() → anime_list_apis.models.MediaData.MangaData

Generates a new MangaData object from the internal representation :return: The generated MangaData object :raises TypeError: If any of the internal parameters has a wrong type

get_user_data() → anime_list_apis.models.MediaUserData.MangaUserData

Generates a new MangaUserData object from the internal representation :return: The generated MangaUserData object :raises TypeError: If any of the internal parameters has a wrong type

class anime_list_apis.models.MediaListEntry.**MediaListEntry**(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *media_data: anime_list_apis.models.MediaData.MediaData*, *user_data: anime_list_apis.models.MediaUserData.MediaUserData*)

Bases: *anime_list_apis.models.Serializable.MediaSerializable*, *anime_list_apis.models.CacheAble.CacheAble*

Class that models a user's media list entry

__init__(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*, *media_data: anime_list_apis.models.MediaData.MediaData*, *user_data: anime_list_apis.models.MediaUserData.MediaUserData*)

Initializes the media list entry. :param media_data: The media data :param user_data: The user data :raises TypeError: If any of the parameters has a wrong type :raises ValueError: If the media and user data do not match

classmethod get_class_for_media_type(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*)
Maps a class to a media type :param media_type: The media type :return: The class mapped to that media type

get_id() → anime_list_apis.models.attributes.Id.Id
Retrieves the cache entry's ID :return: The ID

get_media_data() → anime_list_apis.models.MediaData.MediaData
Generates a new MediaData object from the internal representation :return: The generated MediaData object :raises TypeError: If any of the internal parameters has a wrong type

get_media_type() → anime_list_apis.models.attributes.MediaType.MediaType
Retrieves the media type :return: The media type

get_model_type() → anime_list_apis.models.CacheAble.CacheModelType
Retrieves the cache model type :return: The model type

get_user_data() → anime_list_apis.models.MediaUserData.MediaUserData
Generates a new MediaUserData object from the internal representation :return: The generated MediaUserData object :raises TypeError: If any of the internal parameters has a wrong type

get_username() → Optional[str]
Retrieves the username, if applicable. Else None :return: The username or None if not applicable

is_valid_entry() → bool
Checks if the data has a valid combination of entry data :return: True, if all required attributes are valid and present

anime_list_apis.models.MediaUserData module

class anime_list_apis.models.MediaUserData.**AnimeUserData**(*media_id: anime_list_apis.models.attributes.Id.Id, username: str, score: anime_list_apis.models.attributes.Score.Score, consuming_status: anime_list_apis.models.attributes.ConsumingStatu consuming_start: Optional[anime_list_apis.models.attributes.Date.Da consuming_end: Optional[anime_list_apis.models.attributes.Date.Da episode_progress: int*)
Bases: *anime_list_apis.models.MediaUserData.MediaUserData*

Models a user's entry data for an anime series

__init__(*media_id: anime_list_apis.models.attributes.Id.Id, username: str, score: anime_list_apis.models.attributes.Score.Score, consuming_status: anime_list_apis.models.attributes.ConsumingStatus.ConsumingStatus, consuming_start: Optional[anime_list_apis.models.attributes.Date.Date], consuming_end: Optional[anime_list_apis.models.attributes.Date.Date], episode_progress: int*)
Initializes the AnimeUserData object :param media_id: The ID of the corresponding AnimeData object :param username: The user's username :param score: The user's score for this anime :param consuming_status: The user's current watching status :param consuming_start: The date on which the user started watching :param consuming_end: The date on which the user completed the show :param episode_progress: The user's progress :raises TypeError: If any of the parameters has a wrong type

class anime_list_apis.models.MediaUserData.**MangaUserData**(*media_id: anime_list_apis.models.attributes.Id.Id, username: str, score: anime_list_apis.models.attributes.Score.Score, consuming_status: anime_list_apis.models.attributes.ConsumingStatu consuming_start: Optional[anime_list_apis.models.attributes.Date.Da consuming_end: Optional[anime_list_apis.models.attributes.Date.Da chapter_progress: int, volume_progress: int*)

Bases: *anime_list_apis.models.MediaUserData.MediaUserData*

Models a user's entry data for a manga series

__init__(*media_id: anime_list_apis.models.attributes.Id.Id, username: str, score: anime_list_apis.models.attributes.Score.Score, consuming_status: anime_list_apis.models.attributes.ConsumingStatus.ConsumingStatus, consuming_start: Optional[anime_list_apis.models.attributes.Date.Date], consuming_end: Optional[anime_list_apis.models.attributes.Date.Date], chapter_progress: int, volume_progress: int*)

Initializes the MangaUserData object :param media_id: The ID of the corresponding MangaData object :param username: The user's username :param score: The user's score for this manga :param consuming_status: The user's current reading status :param consuming_start: The date on which the user started reading :param consuming_end: The date on which the user completed the series :param chapter_progress: The user's chapter progress :param volume_progress: The user's volume progress :raises TypeError: If any of the parameters has a wrong type

class anime_list_apis.models.MediaUserData.**MediaUserData**(*media_id: anime_list_apis.models.attributes.Id.Id, media_type: anime_list_apis.models.attributes.MediaType.Med username: str, score: anime_list_apis.models.attributes.Score.Score, consuming_status: anime_list_apis.models.attributes.ConsumingStatu consuming_start: Optional[anime_list_apis.models.attributes.Date.Da consuming_end: Optional[anime_list_apis.models.attributes.Date.Da*

Bases: *anime_list_apis.models.Serializable.MediaSerializable*, *anime_list_apis.models.CacheAble.CacheAble*

Models a user's entry data for an anime series

__init__(*media_id: anime_list_apis.models.attributes.Id.Id, media_type: anime_list_apis.models.attributes.MediaType.MediaType, username: str, score: anime_list_apis.models.attributes.Score.Score, consuming_status: anime_list_apis.models.attributes.ConsumingStatus.ConsumingStatus, consuming_start: Optional[anime_list_apis.models.attributes.Date.Date], consuming_end: Optional[anime_list_apis.models.attributes.Date.Date]*)

Initializes the MediaUserData object :param media_id: The ID of the corresponding MediaData object :param media_type: The type of media represented :param username: The user's username :param score: The user's score for this entry :param consuming_status: The user's current consuming status :param consuming_start: The date on which the user started consuming :param consuming_end: The date on which the user completed the entry :raises TypeError: If any of the parameters has a wrong type

classmethod get_class_for_media_type(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*)
Maps a class to a media type :param media_type: The media type :return: The class mapped to that media type

get_id() → anime_list_apis.models.attributes.Id.Id
Retrieves the cache entry's ID :return: The ID

get_media_type() → anime_list_apis.models.attributes.MediaType.MediaType
Retrieves the media type :return: The media type

get_model_type() → anime_list_apis.models.CacheAble.CacheModelType
Retrieves the cache model type :return: The model type

get_username() → Optional[str]
Retrieves the username, if applicable. Else None :return: The username or None if not applicable

is_valid_entry() → bool
Checks if the data has a valid combination of entry data :return: True, if all required attributes are valid and present

anime_list_apis.models.Serializable module

class anime_list_apis.models.Serializable.**MediaSerializable**
Bases: *anime_list_apis.models.Serializable.Serializable*

Class that allows for easier subclassing of Media classes

classmethod get_class_for_media_type(*media_type: anime_list_apis.models.attributes.MediaType.MediaType*)
Maps a class to a media type :param media_type: The media type :return: The class mapped to that media type

class anime_list_apis.models.Serializable.**Serializable**
Bases: object

Abstract class that defines methods for subclasses to implement to make sure that they can be serialized

classmethod deserialize(*data: Dict[str, Optional[str]]*)
Deserializes a dictionary into an object of this type :param data: The data to deserialize :return: The deserialized object :raises TypeError: If a type error occurred :raises ValueError: If the data could not be deserialized

classmethod ensure_type(*obj: object*, *typ: type*, *none_allowed: bool = False*)
Raises a TypeError if the object does not match the type :param obj: The object to check :param typ: The type the object should be :param none_allowed: If True, allows None values :return: None :raises TypeError: If the types do not match

serialize() → Dict[str, Optional[str]]
Serializes the object into a dictionary :return: The serialized form of this object

static type_check(*obj: object*, *typ: type*, *none_allowed: bool = False*) → bool
Checks if an object is an instance of a type :param obj: The object to check :param typ: The type it should be :param none_allowed: If True, allows None values :return: True if the object is of that type, else False

Module contents

anime_list_apis.test package

Subpackages

anime_list_apis.test.api package

Submodules

anime_list_apis.test.api.TestAnilistApi module

class anime_list_apis.test.api.TestAnilistApi.**TestAnilistApi**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the Anilist Api. May be subclassed to test other APIs as well

api_class
alias of *anime_list_apis.api.AnilistApi.AnilistApi*

setUp()
Creates a cache :return: None

tearDown()
Removes all generated files and directories :return: None

test_caching_anime()
Tests that the caching works correctly for media data :return: None

test_checking_if_in_list()
Tests the is_in_list() method :return: None

test_fetching_related_data()
Tests fetching data for related data :return: None

test_fetching_with_invalid_id_type()
Makes sure that trying to use an unsupported ID type results in a None object :return: None

test_getting_fresh_data()
Tests retrieving fresh data :return: None

test_retrieving_data()
Tests retrieving a data, user data and list entries for both anime and manga :return: None

test_retrieving_entry_for_non_existant_user()
Tests retrieving an entry for a non-existant user :return: None

test_retrieving_invalid_entry()
Tests retrieving invalid entries Assumptions: No ids < 0 or > 1000000000 :return: None

test_retrieving_list_for_nonexistant_user()
Tests retrieving an anime list for an invalid user :return: None

test_retrieving_lists()
Tests retrieving a user's lists using various methods :return: None

test_retrieving_non_existant_list_entry()
Tests retrieving entries that are not in a user's list Uses very badly rated entries to make sure that they never get added to the list, breaking the test :return: None

valid_id_types = [<IdType.ANILIST: 2>, <IdType.MYANIMELIST: 1>]
A list of valid ID types

class anime_list_apis.test.api.TestAnilistApi.**TestAnilistApiSpecific**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests specifically for the Anilist API

setUp()
Creates a cache :return: None

tearDown()
Removes all generated files and directories :return: None

test_filling_in_english_title_with_romaji()
Some titles don't have an english title entry on anilist. In such cases, the romaji will automatically be filled in into the english title. :return: None

test_getting_anilist_id_from_mal_id()
Tests retrieving an anilist ID for a myanimelist ID :return: None

test_getting_anilist_info_with_invalid_mal_id()
Tests retrieving anilist data with an invalid myanimelist ID :return: None

Module contents

anime_list_apis.test.cache package

Submodules

anime_list_apis.test.cache.TestCache module

class anime_list_apis.test.cache.TestCache.**TestCacher**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the Cacher

setUp()
Creates a cache :return: None

tearDown()
Removes all generated files and directories :return: None

test_autowrite()
Tests the automatic writing of the cache after x amount of added entries :return: None

test_caching_entry()
Tests caching and retrieving a Media List Entry :return: None

test_caching_primitive_data()
Tests caching primitive data :return: None

test_delayed_autowrite()
Tests that cache data is written to file automatically after a certain amount of added entries :return: None

test_generating_new_cache()
Tests creating a new cache in a custom location :return: None

test_getting_from_other_site_type()
Tests that it's not possible to get an entry from another site :return: None

test_invalidating_cache()
Tests invalidating cache entries :return: None

test_lifetime()
Tests the lifetime of cache entries :return: None

test_loading_and_retrieving_cache()
Tests writing some data into the cache and then reloading it in a different Cache object :return: None

test_not_being_able_to_fetch_incomplete_list_entry()
Makes sure that one can't fetch an incomplete media list entry :return: None

test_reloading()
Tests reloading the cache :return: None

test_retrieving_non_existant_data()
Tests retrieving an entry from the cache that doesn't exist :return: None

test_using_int_or_id_object()
Tests retrieving entries by using ints and Id objects interchangeably :return: None

Module contents

anime_list_apis.test.models package

Subpackages

anime_list_apis.test.models.attributes package

Submodules

anime_list_apis.test.models.attributes.TestDate module

class anime_list_apis.test.models.attributes.TestDate.**TestDate**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the Date Attribute class

test_deserialization()
Tests deserializing a Date object :return: None

test_equality()
Tests that the equality of the objects is handled correctly :return: None

test_invalid_constructor_parameters()
Tests using invalid parameter types with the constructor :return: None

test_invalid_deserialization()
Tests that invalid serialized data raises ValueErrors when deserialized :return: None

test_serialization()
Tests serializing a Date object :return: None

test_string_representation()
Tests that the string representation is correct :return: None

anime_list_apis.test.models.attributes.TestId module

class anime_list_apis.test.models.attributes.TestId.**TestId**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the Id Attribute class

test_deserialization()
Tests deserializing an ID object :return: None

test_equality()
Tests that the equality of the objects is handled correctly :return: None

test_fetching_different_id_types()
Tests generating an ID using the constructor and fetching the different IDs :return: None

test_invalid_constructor_parameters()
Tests using invalid parameter types with the constructor :return: None

test_invalid_deserialization()
Tests that invalid serialized data raises ValueErrors when deserialized :return: None

test_no_entries()
Tests that the constructor raises a ValueError when no ID at all is provided :return: None

test_no_valid_entries()
Tests that providing None values as the only IDs still result in a ValueError :return: None

test_serialization()
Tests serializing an ID object :return: None

test_setting_ids()
Tests manually setting IDs after construction :return: None

test_setting_invalid_ids()
Tests setting ids that are invalid types :return: None

test_string_representation()
Tests that the string representation is correct :return: None

test_unfilled_entries()
Tests that unfilled ID entries lead to Null values for the respective IDs :return: None

anime_list_apis.test.models.attributes.TestRelation module

class anime_list_apis.test.models.attributes.TestRelation.**TestRelation**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the Relation Attribute class

test_deserialization()
Tests deserializing an ID object :return: None

test_equality()
Tests that the equality of the objects is handled correctly :return: None

test_important_relations()
Tests if relations are correctly identified as “important” :return: None

test_invalid_constructor_parameters()
Tests using invalid parameter types with the constructor :return: None

test_invalid_deserialization()
Tests that invalid serialized data raises ValueErrors when deserialized :return: None

test_serialization()
Tests serializing a Relation object :return: None

test_string_representation()
Tests that the string representation is correct :return: None

test_using_same_ids()
Makes sure that using the same ID for both the source and the destination results in an error :return: None

anime_list_apis.test.models.attributes.TestScore module

class anime_list_apis.test.models.attributes.TestScore.**TestScore**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the Score Attribute class

test_deserialization()
Tests deserializing an ID object :return: None

test_equality()
Tests that the equality of the objects is handled correctly :return: None

test_initializing_different_score_types()
Tests initializing all types of scores with equivalent :return:

test_invalid_constructor_parameters()
Tests using invalid parameter types with the constructor :return: None

test_invalid_deserialization()
Tests that invalid serialized data raises ValueErrors when deserialized :return: None

test_invalid_score()
Tests using an invalid score :return: None

test_permanent_conversion()
Tests converting the internal score to another score type :return: None

test_score_calculation()
Tests calculating different score types- :return: None

test_serialization()
Tests serializing and deserializing an ID object :return: None

test_string_representation()
Tests that the string representation is correct :return: None

anime_list_apis.test.models.attributes.TestTitle module

class anime_list_apis.test.models.attributes.TestTitle.**TestTitle**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the Title Attribute class

test_automatically_using_different_default_title_type()
Tests changing the default title type by not supplying the default title type. :return: None

test_changing_default_invalid_title_type()
Tests changing the default title type with an invalid title type :return: None

test_changing_default_title_type()
Tests changing the default title type :return: None

test_deserialization()
Tests deserializing an ID object :return: None

test_different_default_title_type()
Tests using a different title type :return: None

test_equality()
Tests that the equality of the objects is handled correctly :return: None

test_invalid_constructor_parameters()
Tests using invalid parameter types with the constructor :return: None

test_invalid_deserialization()
Tests that invalid serialized data raises ValueErrors when deserialized :return: None

test_missing_entries()
Tests that missing title entries are replaced with None. :return: None

test_no_entries()
Tests that at least on title is required during initialization :return: None

test_serialization()
Tests serializing an ID object :return: None

test_setting_titles()
Tests manually setting title values :return: None

test_setting_titles_with_invalid_types()
Tests that invalid types in title setting parameters raise a TypeError :return: None

test_string_representation()
Tests that the string representation is correct :return: None

test_title_construction()
Tests constructing the title :return: None

Module contents

Submodules

anime_list_apis.test.models.TestCacheAble module

class anime_list_apis.test.models.TestCacheAble.**TestCacheAble**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Class that tests cache-able objects

test_getters()
Tests if the getter methods work correctly :return:

test_tag_generation()
Tests generating tags :return: None

anime_list_apis.test.models.TestMediaData module

class anime_list_apis.test.models.TestMediaData.**TestMediaData**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the MediaData Model class

static generate_sample_anime_data() → anime_list_apis.models.MediaData.AnimeData
Generates a generic AnimeData object :return: The generated anime data object

static generate_sample_manga_data() → anime_list_apis.models.MediaData.MangaData
Generates a generic MangaData object :return: The generated manga data object

static generate_sample_serialized_anime_data() → Dict[str, Optional[str]]
Generates some sample serialized anime data :return: The serialized sample data

static generate_sample_serialized_manga_data() → Dict[str, Optional[str]]
Generates some sample serialized manga data :return: The serialized sample data

test_deserialization()
Tests deserializing an MediaData object :return: None

test_equality()
Tests that the equality of the objects is handled correctly :return: None

test_generating_anime_data()
Tests generating an anime data object :return: None

test_generating_manga_data()
Tests generating a manga data object :return: None

test_generating_with_none_value()
Tests using a valid None value in the constructor :return: None

test_generic_deserialization()
Tests using the MediaData class to deserialize :return: None

test_invalid_deserialization()
Tests that invalid serialized data raises ValueErrors when deserialized :return: None

test_serialization()
Tests serializing an MediaData object :return: None

test_string_representation()
Tests that the string representation is correct :return: None

anime_list_apis.test.models.TestMediaListEntry module

class anime_list_apis.test.models.TestMediaListEntry.**TestMediaListEntry**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the MediaListEntry Model class

static generate_sample_anime_entry() → anime_list_apis.models.MediaListEntry.AnimeListEntry
Generates a sample AnimeListEntry object :return: The generated AnimeListEntry object

static generate_sample_manga_entry() → anime_list_apis.models.MediaListEntry.MangaListEntry
Generates a sample MangaListEntry object :return: The generated MangaListEntry object

static generate_sample_serialized_anime_entry() → Dict[str, Optional[str]]
Generates a sample deserialized version of the sample anime entry :return: The deserialized sample entry

static generate_sample_serialized_manga_entry() → Dict[str, Optional[str]]
Generates a sample deserialized version of the sample manga entry :return: The deserialized sample entry

test_deserialization()
Tests deserializing a MediaUserData object :return: None

test_equality()
Tests that the equality of the objects is handled correctly :return: None

test_generating_anime_media_list_entry()
Tests generating an anime media list entry object :return: None

test_generating_manga_media_list_entry()
Tests generating a manga media list entry object :return: None

test_generic_deserialization()
Tests using the MediaListEntry class to deserialize :return: None

test_if_valid()
Tests the functionality of the is_valid_entry method :return: None

test_internal_getters()
Tests the generating getter methods that generate MediaData and MediaUserData objects :return: None

test_invalid_deserialization()
Tests that invalid serialized data raises ValueErrors when deserialized :return: None

test_mismatching_ids()
Tests that two mismatched IDs are identified and raise a ValueError :return: None

test_mixing_anime_and_manga()
Tests that it's impossible to mix anime and manga data :return:

test_serialization()
Tests serializing a MediaUserData object :return: None

test_string_representation()
Tests that the string representation is correct :return: None

anime_list_apis.test.models.TestMediaUserData module

class anime_list_apis.test.models.TestMediaUserData.**TestMediaUserData**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests the MediaUserData Model class

static generate_sample_anime_user_data() → anime_list_apis.models.MediaUserData.AnimeUserData
Generates a sample AnimeUserData object :return: The generated AnimeUserData object

static generate_sample_manga_user_data() → anime_list_apis.models.MediaUserData.MangaUserData
Generates a sample MangaUserData object :return: The generated MangaUserData object

static generate_sample_serialized_anime_user_data() → Dict[str, Optional[str]]
Generates a sample serialized AnimeUserData object :return: The serialized data

static generate_sample_serialized_manga_user_data() → Dict[str, Optional[str]]
Generates a sample serialized MangaUserData object :return: The serialized data

test_deserialization()
Tests deserializing a MediaUserData object :return: None

test_equality()
Tests that the equality of the objects is handled correctly :return: None

test_generating_anime_user_entry_data()
Tests generating an anime user entry data object :return: None

test_generating_manga_user_entry_data()
Tests generating an manga user entry data object :return: None

test_generic_deserialization()
Tests using the MediaUserData class to deserialize :return: None

test_invalid_deserialization()
Tests that invalid serialized data raises ValueErrors when deserialized :return: None

test_none_parameters()
Tests using None as parameters :return: None

test_serialization()
Tests serializing a MediaUserData object :return: None

test_string_representation()
Tests that the string representation is correct :return: None

test_valid()
Makes sure that it's possible to check if an entry is valid :return: None

Module contents

Module contents

1.1.2 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

anime_list_apis, 28
anime_list_apis.api, 6
anime_list_apis.api.AnilistApi, 3
anime_list_apis.api.ApiInterface, 3
anime_list_apis.api.KitsuApi, 5
anime_list_apis.api.MyanimelistApi, 6
anime_list_apis.cache, 8
anime_list_apis.cache.Cache, 6
anime_list_apis.models, 20
anime_list_apis.models.attributes, 13
anime_list_apis.models.attributes.ConsumingStatus, 8
anime_list_apis.models.attributes.Date, 9
anime_list_apis.models.attributes.Id, 9
anime_list_apis.models.attributes.MediaFormat, 9
anime_list_apis.models.attributes.MediaType, 10
anime_list_apis.models.attributes.Relation, 10
anime_list_apis.models.attributes.ReleasingStatus, 11
anime_list_apis.models.attributes.Score, 11
anime_list_apis.models.attributes.Title, 12
anime_list_apis.models.CacheAble, 13
anime_list_apis.models.MediaData, 13
anime_list_apis.models.MediaListEntry, 16
anime_list_apis.models.MediaUserData, 17
anime_list_apis.models.Serializable, 19
anime_list_apis.test, 28
anime_list_apis.test.api, 21
anime_list_apis.test.api.TestAnilistApi, 20
anime_list_apis.test.cache, 22
anime_list_apis.test.cache.TestCache, 21
anime_list_apis.test.models, 28
anime_list_apis.test.models.attributes, 25
anime_list_apis.test.models.attributes.TestDate, 22
anime_list_apis.test.models.attributes.TestId, 23
anime_list_apis.test.models.attributes.TestRelation, 23
anime_list_apis.test.models.attributes.TestScore, 24
anime_list_apis.test.models.attributes.TestTitle, 24
anime_list_apis.test.models.TestCacheAble, 25
anime_list_apis.test.models.TestMediaData, 26
anime_list_apis.test.models.TestMediaListEntry, 26
anime_list_apis.test.models.TestMediaUserData, 27

INDEX

Symbols

__init__() (*anime_list_apis.api.AnilistApi.AnilistApi method*), 3
__init__() (*anime_list_apis.api.ApiInterface.ApiInterface method*), 3
__init__() (*anime_list_apis.api.KitsuApi.KitsuApi method*), 6
__init__() (*anime_list_apis.api.MyanimelistApi.MyanimelistApi method*), 6
__init__() (*anime_list_apis.cache.Cache.Cache method*), 6
__init__() (*anime_list_apis.models.MediaData.AnimeData method*), 14
__init__() (*anime_list_apis.models.MediaData.MangaData method*), 14
__init__() (*anime_list_apis.models.MediaData.MediaData method*), 15
__init__() (*anime_list_apis.models.MediaListEntry.AnimeListEntry method*), 16
__init__() (*anime_list_apis.models.MediaListEntry.MangaListEntry method*), 16
__init__() (*anime_list_apis.models.MediaListEntry.MediaListEntry method*), 16
__init__() (*anime_list_apis.models.MediaUserData.AnimeUserData method*), 17
__init__() (*anime_list_apis.models.MediaUserData.MangaUserData method*), 18
__init__() (*anime_list_apis.models.MediaUserData.MediaUserData method*), 18
__init__() (*anime_list_apis.models.attributes.Date.Date method*), 9
__init__() (*anime_list_apis.models.attributes.Id.Id method*), 9
__init__() (*anime_list_apis.models.attributes.Relation.Relation method*), 10
__init__() (*anime_list_apis.models.attributes.Score.Score method*), 11
__init__() (*anime_list_apis.models.attributes.Title.Title method*), 12

A

ADAPTATION (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 10
add() (*anime_list_apis.cache.Cache.Cache method*), 6
add_primitive() (*anime_list_apis.cache.Cache.Cache method*), 7
ALTERNATIVE (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11
ANILIST (*anime_list_apis.models.attributes.Id.IdType attribute*), 9
AnilistApi (*class in anime_list_apis.api.AnilistApi*), 3
ANIME (*anime_list_apis.models.attributes.MediaType.MediaType attribute*), 10
anime_list_apis (*module*), 28
anime_list_apis.api (*module*), 6
anime_list_apis.api.AnilistApi (*module*), 3
anime_list_apis.api.ApiInterface (*module*), 3
anime_list_apis.api.KitsuApi (*module*), 5
anime_list_apis.api.MyanimelistApi (*module*), 6
anime_list_apis.cache (*module*), 8
anime_list_apis.cache.Cache (*module*), 6
anime_list_apis.models (*module*), 20
anime_list_apis.models.attributes (*module*), 13
anime_list_apis.models.attributes.ConsumingStatus (*module*), 8
anime_list_apis.models.attributes.Date (*module*), 9
anime_list_apis.models.attributes.Id (*module*), 9
anime_list_apis.models.attributes.MediaFormat (*module*), 9
anime_list_apis.models.attributes.MediaType (*module*), 10
anime_list_apis.models.attributes.Relation (*module*), 10
anime_list_apis.models.attributes.ReleasingStatus (*module*), 11
anime_list_apis.models.attributes.Score (*module*), 11
anime_list_apis.models.attributes.Title

(*module*), 12
anime_list_apis.models.CacheAble (*module*), 13
anime_list_apis.models.MediaData (*module*), 13
anime_list_apis.models.MediaListEntry (*module*), 16
anime_list_apis.models.MediaUserData (*module*), 17
anime_list_apis.models.Serializable (*module*), 19
anime_list_apis.test (*module*), 28
anime_list_apis.test.api (*module*), 21
anime_list_apis.test.api.TestAnilistApi (*module*), 20
anime_list_apis.test.cache (*module*), 22
anime_list_apis.test.cache.TestCache (*module*), 21
anime_list_apis.test.models (*module*), 28
anime_list_apis.test.models.attributes (*module*), 25
anime_list_apis.test.models.attributes.TestDate (*module*), 22
anime_list_apis.test.models.attributes.TestId (*module*), 23
anime_list_apis.test.models.attributes.TestRelation (*module*), 23
anime_list_apis.test.models.attributes.TestScore (*module*), 24
anime_list_apis.test.models.attributes.TestTitle (*module*), 24
anime_list_apis.test.models.TestCacheAble (*module*), 25
anime_list_apis.test.models.TestMediaData (*module*), 26
anime_list_apis.test.models.TestMediaListEntry (*module*), 26
anime_list_apis.test.models.TestMediaUserData (*module*), 27
AnimeData (*class in anime_list_apis.models.MediaData*), 13
AnimeListEntry (*class in anime_list_apis.models.MediaListEntry*), 16
AnimeUserData (*class in anime_list_apis.models.MediaUserData*), 17
api_class (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi attribute*), 20
ApiInterface (*class in anime_list_apis.api.ApiInterface*), 3

C

Cache (*class in anime_list_apis.cache.Cache*), 6
CacheAble (*class in anime_list_apis.models.CacheAble*), 13
CacheModelType (*class in anime_list_apis.models.CacheAble*), 13
CANCELLED (*anime_list_apis.models.attributes.ReleasingStatus.Releasing attribute*), 11
change_default_title_type() (*anime_list_apis.models.attributes.Title.Title method*), 12
CHARACTER (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11
COMPLETED (*anime_list_apis.models.attributes.ConsumingStatus.Consum attribute*), 8
ConsumingStatus (*class in anime_list_apis.models.attributes.ConsumingStatus*), 8
convert() (*anime_list_apis.models.attributes.Score.Score method*), 11
CURRENT (*anime_list_apis.models.attributes.ConsumingStatus.Consuming attribute*), 8

D

DATA (*anime_list_apis.models.CacheAble.CacheModelType attribute*), 13
Date (*class in anime_list_apis.models.attributes.Date*), 9
deserialize() (*anime_list_apis.models.Serializable.Serializable class method*), 19
DROPPED (*anime_list_apis.models.attributes.ConsumingStatus.Consuming attribute*), 8

E

ENGLISH (*anime_list_apis.models.attributes.Title.TitleType attribute*), 12
ensure_type() (*anime_list_apis.models.Serializable.Serializable class method*), 19

F

FINISHED (*anime_list_apis.models.attributes.ReleasingStatus.ReleasingS attribute*), 11
FIVE_POINT (*anime_list_apis.models.attributes.Score.ScoreType attribute*), 12

G

generate_id_tag() (*anime_list_apis.cache.Cache.Cache static method*), 7
generate_sample_anime_data() (*anime_list_apis.test.models.TestMediaData.TestMediaData static method*), 26
generate_sample_anime_entry() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEn static method*), 26

generate_sample_anime_user_data() (anime_list_apis.test.models.TestMediaUserData.TestMediaUserData static method), 27
generate_sample_manga_data() (anime_list_apis.test.models.TestMediaData.TestMediaData static method), 26
generate_sample_manga_entry() (anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry static method), 26
generate_sample_manga_user_data() (anime_list_apis.test.models.TestMediaUserData.TestMediaUserData static method), 27
generate_sample_serialized_anime_data() (anime_list_apis.test.models.TestMediaData.TestMediaData static method), 26
generate_sample_serialized_anime_entry() (anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry static method), 26
generate_sample_serialized_anime_user_data() (anime_list_apis.test.models.TestMediaUserData.TestMediaUserData static method), 27
generate_sample_serialized_manga_data() (anime_list_apis.test.models.TestMediaData.TestMediaData static method), 26
generate_sample_serialized_manga_entry() (anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry static method), 26
generate_sample_serialized_manga_user_data() (anime_list_apis.test.models.TestMediaUserData.TestMediaUserData static method), 27
generate_tag() (anime_list_apis.models.CacheAble.CacheAble method), 13
get() (anime_list_apis.cache.Cache.Cache method), 7
get() (anime_list_apis.models.attributes.Id.Id method), 9
get() (anime_list_apis.models.attributes.Score.Score method), 11
get() (anime_list_apis.models.attributes.Title.Title method), 12
get_anilist_id_from_mal_id() (anime_list_apis.api.AnilistApi.AnilistApi method), 3
get_anime_data() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_anime_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_anime_list_entry() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_anime_user_data() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_anime_user_data_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_class_for_media_type() (anime_list_apis.models.MediaData.MediaData class method), 15
get_class_for_media_type() (anime_list_apis.models.MediaListEntry.MediaListEntry class method), 16
get_class_for_media_type() (anime_list_apis.models.MediaUserData.MediaUserData class method), 18
get_class_for_media_type() (anime_list_apis.models.Serializable.MediaSerializable class method), 19
get_data() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_id() (anime_list_apis.models.CacheAble.CacheAble method), 13
get_id() (anime_list_apis.models.MediaData.MediaData method), 15
get_id() (anime_list_apis.models.MediaListEntry.MediaListEntry method), 17
get_id() (anime_list_apis.models.MediaUserData.MediaUserData method), 19
get_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_list_entry() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_manga_data() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_manga_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 4
get_manga_list_entry() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
get_manga_user_data() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
get_manga_user_data_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
get_media_data() (anime_list_apis.cache.Cache.Cache method), 7
get_media_data() (anime_list_apis.models.MediaListEntry.AnimeList method), 16
get_media_data() (anime_list_apis.models.MediaListEntry.MangaList method), 16
get_media_data() (anime_list_apis.models.MediaListEntry.MediaList method), 17
get_media_list_entry() (anime_list_apis.cache.Cache.Cache method), 7
get_media_type() (anime_list_apis.models.CacheAble.CacheAble method), 13
get_media_type() (anime_list_apis.models.MediaData.MediaData

method), 15
get_media_type() (anime_list_apis.models.MediaListEntry.MediaListEntry method), 17
get_media_type() (anime_list_apis.models.MediaUserData.MediaUserData method), 19
get_media_user_data() (anime_list_apis.cache.Cache.Cache method), 7
get_model_type() (anime_list_apis.models.CacheAble.CacheAble method), 13
get_model_type() (anime_list_apis.models.MediaData.MediaData method), 15
get_model_type() (anime_list_apis.models.MediaListEntry.MediaListEntry method), 17
get_model_type() (anime_list_apis.models.MediaUserData.MediaUserData method), 19
get_primitive() (anime_list_apis.cache.Cache.Cache method), 7
get_related_data() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
get_user_data() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
get_user_data() (anime_list_apis.models.MediaListEntry.AnimeListEntry method), 16
get_user_data() (anime_list_apis.models.MediaListEntry.MangaListEntry method), 16
get_user_data() (anime_list_apis.models.MediaListEntry.MediaListEntry method), 17
get_user_data_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
get_username() (anime_list_apis.models.CacheAble.CacheAble method), 13
get_username() (anime_list_apis.models.MediaData.MediaData method), 15
get_username() (anime_list_apis.models.MediaListEntry.MediaListEntry method), 17
get_username() (anime_list_apis.models.MediaUserData.MediaUserData method), 19

I

Id (class in anime_list_apis.models.attributes.Id), 9
IdType (class in anime_list_apis.models.attributes.Id), 9
invalidate() (anime_list_apis.cache.Cache.Cache method), 7
invalidate_media_data() (anime_list_apis.cache.Cache.Cache method), 7
invalidate_media_list_entry() (anime_list_apis.cache.Cache.Cache method), 8
invalidate_media_user_data() (anime_list_apis.cache.Cache.Cache method), 8
is_important() (anime_list_apis.models.attributes.Relation.Relation method), 10
is_in_anime_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
is_in_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
is_in_manga_list() (anime_list_apis.api.ApiInterface.ApiInterface method), 5
is_valid() (anime_list_apis.models.MediaListEntry.MediaListEntry method), 17
is_valid() (anime_list_apis.models.MediaUserData.MediaUserData method), 19

J

JAPANESE (anime_list_apis.models.attributes.Title.TitleType attribute), 12

K

KITSU (anime_list_apis.models.attributes.Id.IdType attribute), 9
KitsuApi (class in anime_list_apis.api.KitsuApi), 5

L

LIGHT_NOVEL (anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute), 9
load() (anime_list_apis.cache.Cache.Cache method), 8

M

MANGA (anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute), 9
MANGA (anime_list_apis.models.attributes.MediaType.MediaType attribute), 10
MangaData (class in anime_list_apis.models.MediaData), 14
MangaListEntry (class in anime_list_apis.models.MediaListEntry), 16
MangaUserData (class in anime_list_apis.models.MediaUserData), 17
MEDIA_DATA (anime_list_apis.models.CacheAble.CacheModelType attribute), 13
MEDIA_LIST_ENTRY (anime_list_apis.models.CacheAble.CacheModelType attribute), 13
MEDIA_USER_DATA (anime_list_apis.models.CacheAble.CacheModelType attribute), 13
MediaData (class in anime_list_apis.models.MediaData), 14

MediaFormat (*class in anime_list_apis.models.attributes.MediaFormat*), 9
MediaListEntry (*class in anime_list_apis.models.MediaListEntry*), 16
MediaSerializable (*class in anime_list_apis.models.Serializable*), 19
MediaType (*class in anime_list_apis.models.attributes.MediaType*), 10
MediaUserData (*class in anime_list_apis.models.MediaUserData*), 18
model_map (*anime_list_apis.cache.Cache.Cache attribute*), 8
MOVIE (*anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute*), 9
MUSIC (*anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute*), 9
MYANIMELIST (*anime_list_apis.models.attributes.Id.IdType attribute*), 9
MyanimelistApi (*class in anime_list_apis.api.MyanimelistApi*), 6

N

NOT_RELEASED (*anime_list_apis.models.attributes.ReleasingStatus.ReleasingStatus attribute*), 11

O

ONA (*anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute*), 9
ONE_SHOT (*anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute*), 9
OTHER (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11
OVA (*anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute*), 10

P

PARENT (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11
PAUSED (*anime_list_apis.models.attributes.ConsumingStatus.ConsumingStatus attribute*), 8
PERCENTAGE (*anime_list_apis.models.attributes.Score.ScoreType attribute*), 12
PLANNING (*anime_list_apis.models.attributes.ConsumingStatus.ConsumingStatus attribute*), 8
PREQUEL (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11

R

Relation (*class in anime_list_apis.models.attributes.Relation*), 10
RelationType (*class in anime_list_apis.models.attributes.Relation*), 10
RELEASING (*anime_list_apis.models.attributes.ReleasingStatus.ReleasingStatus attribute*), 11
ReleasingStatus (*class in anime_list_apis.models.attributes.ReleasingStatus*), 11
REPEATING (*anime_list_apis.models.attributes.ConsumingStatus.ConsumingStatus attribute*), 8
ROMAJI (*anime_list_apis.models.attributes.Title.TitleType attribute*), 12

S

Score (*class in anime_list_apis.models.attributes.Score*), 11
ScoreType (*class in anime_list_apis.models.attributes.Score*), 12
SEQUEL (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11
Serializable (*class in anime_list_apis.models.Serializable*), 19
serialize() (*anime_list_apis.models.Serializable.Serializable method*), 19
set() (*anime_list_apis.models.attributes.Id.Id method*), 9
set() (*anime_list_apis.models.attributes.Title.Title method*), 12
setUp() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
setUp() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApiSpecific method*), 21
setUp() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
SIDE_STORY (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11
SPECIAL (*anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute*), 10
SPIN_OFF (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11
SUMMARY (*anime_list_apis.models.attributes.Relation.RelationType attribute*), 11

T

tearDown() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
tearDown() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApiSpecific method*), 21
tearDown() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
TEN_POINT (*anime_list_apis.models.attributes.Score.ScoreType attribute*), 12
TEN_POINT_DECIMAL (*anime_list_apis.models.attributes.Score.ScoreType*

attribute), 12
test_automatically_using_different_default_title_type() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 24
test_autowrite() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
test_caching_anime() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_caching_entry() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
test_caching_primitive_data() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
test_changing_default_invalid_title_type() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 24
test_changing_default_title_type() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_checking_if_in_list() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_delayed_autowrite() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
test_deserialization() (*anime_list_apis.test.models.attributes.TestDate.TestDate method*), 22
test_deserialization() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_deserialization() (*anime_list_apis.test.models.attributes.TestRelation.TestRelation method*), 23
test_deserialization() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_deserialization() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_deserialization() (*anime_list_apis.test.models.TestMediaData.TestMediaData method*), 26
test_deserialization() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_deserialization() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 27
test_different_default_title_type() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_equality() (*anime_list_apis.test.models.attributes.TestDate.TestDate method*), 22
test_equality() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_equality() (*anime_list_apis.test.models.attributes.TestRelation.TestRelation method*), 23
test_equality() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_equality() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_equality() (*anime_list_apis.test.models.TestMediaData.TestMediaData method*), 26
test_equality() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_equality() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 27
test_fetching_different_id_types() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_fetching_related_data() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_fetching_with_invalid_id_type() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_filling_in_english_title_with_romaji() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApiSpecific method*), 21
test_generating_anime_data() (*anime_list_apis.test.models.TestMediaData.TestMediaData method*), 26
test_generating_anime_media_list_entry() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_generating_anime_user_entry_data() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 28
test_generating_manga_data() (*anime_list_apis.test.models.TestMediaData.TestMediaData method*), 26
test_generating_manga_media_list_entry() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_generating_manga_user_entry_data() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 28
test_generating_new_cache() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
test_generating_with_none_value() (*anime_list_apis.test.models.TestMediaData.TestMediaData method*), 26
test_generic_deserialization() (*anime_list_apis.test.models.TestMediaData.TestMediaData*

method), 26
test_generic_deserialization() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_generic_deserialization() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 28
test_getters() (*anime_list_apis.test.models.TestCacheAble.TestCacheAble method*), 25
test_getting_anilist_id_from_mal_id() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApiSpecific method*), 21
test_getting_anilist_info_with_invalid_mal_id() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApiSpecific method*), 21
test_getting_fresh_data() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_getting_from_other_site_type() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
test_if_valid() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_important_relations() (*anime_list_apis.test.models.attributes.TestRelation.TestRelation method*), 23
test_initializing_different_score_types() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_internal_getters() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_invalid_constructor_parameters() (*anime_list_apis.test.models.attributes.TestDate.TestDate method*), 22
test_invalid_constructor_parameters() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_invalid_constructor_parameters() (*anime_list_apis.test.models.attributes.TestRelation.TestRelation method*), 23
test_invalid_constructor_parameters() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_invalid_constructor_parameters() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_invalid_deserialization() (*anime_list_apis.test.models.attributes.TestDate.TestDate method*), 22
test_invalid_deserialization() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_invalid_deserialization() (*anime_list_apis.test.models.attributes.TestRelation.TestRelation method*), 23
test_invalid_deserialization() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_invalid_deserialization() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_invalid_deserialization() (*anime_list_apis.test.models.TestMediaData.TestMediaData method*), 26
test_invalid_deserialization() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_invalid_deserialization() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 28
test_invalid_score() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_invalidating_cache() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 21
test_lifetime() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 22
test_loading_and_retrieving_cache() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 22
test_mismatching_ids() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_missing_entries() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_mixing_anime_and_manga() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_no_entries() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_no_entries() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_no_valid_entries() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_none_parameters() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 28
test_not_being_able_to_fetch_incomplete_list_entry() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 22
test_permanent_conversion() (*anime_list_apis.test.models.attributes.TestScore.TestScore*

method), 24
test_reloading() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 22
test_retrieving_data() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_retrieving_entry_for_non_existant_user() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_retrieving_invalid_entry() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_retrieving_list_for_nonexistant_user() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_retrieving_lists() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_retrieving_non_existant_data() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 22
test_retrieving_non_existant_list_entry() (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi method*), 20
test_score_calculation() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_serialization() (*anime_list_apis.test.models.attributes.TestDate.TestDate method*), 22
test_serialization() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_serialization() (*anime_list_apis.test.models.attributes.TestRelation.TestRelation method*), 24
test_serialization() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_serialization() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_serialization() (*anime_list_apis.test.models.TestMediaData.TestMediaData method*), 26
test_serialization() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_serialization() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 28
test_setting_ids() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_setting_invalid_ids() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_setting_titles() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_setting_titles_with_invalid_types() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_string_representation() (*anime_list_apis.test.models.attributes.TestDate.TestDate method*), 22
test_string_representation() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_string_representation() (*anime_list_apis.test.models.attributes.TestRelation.TestRelation method*), 24
test_string_representation() (*anime_list_apis.test.models.attributes.TestScore.TestScore method*), 24
test_string_representation() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_string_representation() (*anime_list_apis.test.models.TestMediaData.TestMediaData method*), 26
test_string_representation() (*anime_list_apis.test.models.TestMediaListEntry.TestMediaListEntry method*), 27
test_string_representation() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 28
test_tag_generation() (*anime_list_apis.test.models.TestCacheAble.TestCacheAble method*), 25
test_title_construction() (*anime_list_apis.test.models.attributes.TestTitle.TestTitle method*), 25
test_unfilled_entries() (*anime_list_apis.test.models.attributes.TestId.TestId method*), 23
test_using_int_or_id_object() (*anime_list_apis.test.cache.TestCache.TestCacher method*), 22
test_using_same_ids() (*anime_list_apis.test.models.attributes.TestRelation.TestRelation method*), 24
test_valid() (*anime_list_apis.test.models.TestMediaUserData.TestMediaUserData method*), 28
TestAnilistApi (*class in anime_list_apis.test.api.TestAnilistApi*), 20
TestAnilistApiSpecific (*class in anime_list_apis.test.api.TestAnilistApi*), 21

TestCacheAble (*class in anime_list_apis.test.models.TestCacheAble*), 25
TestCacher (*class in anime_list_apis.test.cache.TestCache*), 21
TestDate (*class in anime_list_apis.test.models.attributes.TestDate*), 22
TestId (*class in anime_list_apis.test.models.attributes.TestId*), 23
TestMediaData (*class in anime_list_apis.test.models.TestMediaData*), 26
TestMediaListEntry (*class in anime_list_apis.test.models.TestMediaListEntry*), 26
TestMediaUserData (*class in anime_list_apis.test.models.TestMediaUserData*), 27
TestRelation (*class in anime_list_apis.test.models.attributes.TestRelation*), 23
TestScore (*class in anime_list_apis.test.models.attributes.TestScore*), 24
TestTitle (*class in anime_list_apis.test.models.attributes.TestTitle*), 24
THREE_POINT (*anime_list_apis.models.attributes.Score.ScoreType attribute*), 12
Title (*class in anime_list_apis.models.attributes.Title*), 12
TitleType (*class in anime_list_apis.models.attributes.Title*), 12
TV (*anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute*), 10
TV_SHORT (*anime_list_apis.models.attributes.MediaFormat.MediaFormat attribute*), 10
type_check() (*anime_list_apis.models.Serializable.Serializable static method*), 19

V

valid_id_types (*anime_list_apis.test.api.TestAnilistApi.TestAnilistApi attribute*), 20

W

write() (*anime_list_apis.cache.Cache.Cache method*), 8