Hanyuu-sama Documentation

Release 1.3

R/a/dio

February 27, 2013

CONTENTS

Contents:

CHAPTER

ONE

HANYUU

1.1 hanyuu Package

1.1.1 `hanyuu` Package

1.1.2 `config` Module

hanyuu.config.**get**(**args*, ***kwargs*)
See `ConfigParser.RawConfigParser`

hanyuu.config.**getfloat**(**args*, ***kwargs*)
See `ConfigParser.RawConfigParser`

hanyuu.config.**getint**(**args*, ***kwargs*)
See `ConfigParser.RawConfigParser`

hanyuu.config.**has_option**(**args*, ***kwargs*)
See `ConfigParser.RawConfigParser`

hanyuu.config.**has_section**(**args*, ***kwargs*)
See `ConfigParser.RawConfigParser`

hanyuu.config.**items**(**args*, ***kwargs*)
See `ConfigParser.RawConfigParser`

hanyuu.config.**load_configuration**(*filepaths*)
Creates a new `ConfigParser.SafeConfigParser` and tries parsing all `filepaths` given.

`filepaths` should be a list of filenames.

Returns nothing, instead assigns itself to the global variable *configuration* and abstracts itself by calling `create_abstractions()`

hanyuu.config.**options**(**args*, ***kwargs*)
See `ConfigParser.RawConfigParser`

hanyuu.config.**reload_configuration**()
Creates a new `ConfigParser.SafeConfigParser` and passes it the same filenames as given in the last call to `load_configuration()`.

This effectively 'reloads' the configuration files.

hanyuu.config.**sections**(**args*, ***kwargs*)
See `ConfigParser.RawConfigParser`

1.1.3 `utils` Module

class `hanyuu.utils.`**`Switch`**(*initial*, *timeout=15*)
Bases: `object`

A timed switch. Evaluates truthy if the time has expired, else falsy.

`reset`(*timeout=15*)

1.1.4 Subpackages

abstractions Package

`abstractions` Package

package to abstract the database from the rest of the code.

For users: submodules are grouped by their overarcing thema such as DJ profiles and users in the same submodule, track information in the same submodule, AFK streamer information in the same submodule, etc.

For developers: submodules should be of the grouping type where closely related data structures are placed together in a module.

`tracks` Module

class `hanyuu.abstractions.tracks.`**`Length`**
Bases: `int`

A simple subclass of `int` to support formatting on it without having to know the exact format or value in the rest of the code.

`format`()

Returns unicode A formatted [hh:]mm:nn string of the integer.

exception `hanyuu.abstractions.tracks.`**`NoTrackEntry`**
Bases: `exceptions.Exception`

Raised when a `Song` instance accesses Track only attributes without having an audio file attached to it.

class `hanyuu.abstractions.tracks.`**`Plays`**(*song*, *sequence*)
Bases: `list`

A simple subclass of `list` to support some extra attributes.

This class is returned when you access `Track.plays` and is a collection of play times of the `Track` in question.

The collection contains `datetime.datetime` objects or objects that act the same as such with extra methods (for future additions).

`add`(*time*, *dj=None*)
Adds a played entry to the `Track` object.

The exact time it was played at. :params time: A `datetime.datetime` instance.

The DJ that played this track at the time. :params dj: A `hanyuu.abstractions.users.DJ` instance.

Returns None

Note: It's good practice to add the DJ argument to all the code already.

The current database however ignores this argument.

---

last

Returns The time that last occured.

Return type `datetime.datetime` object.

remove(*time*, *dj=None*)
Removes a played entry from the `Track` object.

The time this was played at. :params time: A `datetime.datetime` instance.

The DJ that played this track at the time. If this is `None` it will be ignored otherwise it will be used for exact matching. :params dj: A `hanyuu.abstractions.users.DJ` instance.

Returns None

Note: Currently the *dj* argument is completely ignored.

---

save()
Saves changes to the database.

class `hanyuu.abstractions.tracks.`**Requests**
Bases: `list`

A simple subclass of `list` to support some extra attributes.

last

Returns The time that last occured.

Return type `datetime.datetime` object.

class `hanyuu.abstractions.tracks.`**Track**(*meta*, ***kwargs*)
Bases: `object`

An instance of a known track in our database. This can also be used for adding new tracks.

A 'known' track is one we have seen before. This means there is no difference between tracks we have an audio file of and ones we only know metadata of. The object easily allows you to check if it has a corresponding audio file available or not.

artist

Returns The artist of this song.

Return type unicode

filename

Returns unicode The filename of the audio file.

Raises `NoTrackEntry` if the song has no audio file.

Note: This is relative to the configured *media.directory* configuration.

---

filepath

Returns unicode The full path to the audio file.

Raises NoTrackEntry if the song has no audio file.

classmethod from_esong_id(*id*)
Returns an instance based on the *esong* table ID.

> **Warning:** Don't use this method in production code.

classmethod from_track_id(*id*)
Returns an instance based on the *tracks* table ID.

> **Warning:** Don't use this method in production code.

length

Returns Length of the song or 0 if none available.

Return type Length

Note: The song length is only 100% accurate if the song has an audio file available. Otherwise it's an approximation from when it was last played.

---

metadata

Returns A metadata string of '[artist -] title' where artist is optional

Return type unicode

Note: This uses the *tracks* table if available before trying the other table.

---

open(**args*, ***kwargs*)
Opens the associated file and returns a file object.

This handles the path finding for you.

Params unicode mode The mode to be passed to the open() call.

Returns An open file object.

Raises NoTrackEntry if the song has no audio file.

plays

Returns A mutable object with all the playing data in it.

Return type Plays

requests

Returns A mutable objects with all request data in it.

Return type Requests

Raises NoTrackEntry if the song has no audio file.

save()
Saves all the changes done so far on this object into the database.

Note: This method can do multiple queries to the database depending on the changes done on the object.

---

title

Returns The title of this song.

Return type unicode

hanyuu.abstractions.tracks.**create_metadata_string**(*track*)
Creates a '[artist -] title' string of the hanyuu.db.models.Track instance.

hanyuu.abstractions.tracks.**requires_track**(*func*)
Decorator that raises NoTrackEntry if the song instance has no associated audio file in the database.

Currently this only checks if *self._track* is falsy.

users Module

A module used for the abstractions of the users part of the database.

class hanyuu.abstractions.users.**DJ**(*id=None*, *name=None*)
Bases: object

Encapsulates the concept of a DJ in our system.

This abstracts the database from the rest of the code. But does return a database related object since it's a simple one.

classmethod resolve_id(*id*)
Resolves a DJ identifier to a DJ username.

Returns a class instance or raises DoesNotExist

classmethod resolve_name(*name*)
Resolves a DJ username to a DJ identifier.

Returns an integer that is the DJ identifier or 0 if the DJ username does not exist.

db Package

db Package

common Module

legacy Module

models Module

class hanyuu.db.models.**Base**(**args*, ***kwargs*)
Bases: peewee.Model

Simple base class to inherit from so all the other models inherit the database connection used.

DoesNotExist
alias of BaseDoesNotExist

id = <peewee.PrimaryKeyField object at 0x24a3fd0>

class hanyuu.db.models.**DJ**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *djs* table.

DoesNotExist
alias of DJDoesNotExist

css = <peewee.CharField object at 0x24b52d0>

description = <peewee.TextField object at 0x24b51d0>

id = <peewee.PrimaryKeyField object at 0x24b50d0>

image = <peewee.TextField object at 0x24b5210>

name = <peewee.CharField object at 0x24b5190>

priority = <peewee.IntegerField object at 0x24b5290>

queue

user

visible = <peewee.IntegerField object at 0x24b5250>

class hanyuu.db.models.**Fave**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *efave* table.

DoesNotExist
alias of FaveDoesNotExist

id = <peewee.PrimaryKeyField object at 0x24b85d0>

nickname = <peewee.ForeignKeyField object at 0x24b8790>

song = <peewee.ForeignKeyField object at 0x24b87d0>

class hanyuu.db.models.**LastFm**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *lastfm* table.

DoesNotExist
alias of LastFmDoesNotExist

id = <peewee.PrimaryKeyField object at 0x24b5b10>

nick = <peewee.CharField object at 0x24b5bd0>

username = <peewee.CharField object at 0x24b5c10>

class hanyuu.db.models.**NickRequest**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *nickrequesttime* table.

DoesNotExist
alias of NickRequestDoesNotExist

host = <peewee.TextField object at 0x24b5950>

id = <peewee.PrimaryKeyField object at 0x24b5710>

time = <peewee.DateTimeField object at 0x24b5990>

class hanyuu.db.models.**Nickname**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *enick* table.

DoesNotExist
alias of NicknameDoesNotExist

authcode = <peewee.CharField object at 0x24b5ed0>

dtb = <peewee.DateTimeField object at 0x24b5e90>

faves

first_seen = <peewee.DateTimeField object at 0x24b5e50>

id = <peewee.PrimaryKeyField object at 0x24b5c90>

nickname = <peewee.CharField object at 0x24b5e10>

class hanyuu.db.models.**Play**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *eplay* table.

DoesNotExist
alias of PlayDoesNotExist

id = <peewee.PrimaryKeyField object at 0x24b8310>

song = <peewee.ForeignKeyField object at 0x24b8510>

time = <peewee.DateTimeField object at 0x24b8550>

class hanyuu.db.models.**Queue**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the new design *queue* table.

DoesNotExist
alias of QueueDoesNotExist

dj = <peewee.ForeignKeyField object at 0x24b9410>

id = <peewee.PrimaryKeyField object at 0x24b8e10>

ip = <peewee.TextField object at 0x24b93d0>

song = <peewee.ForeignKeyField object at 0x24b9350>

time = <peewee.DateTimeField object at 0x24b9310>

track = <peewee.ForeignKeyField object at 0x24b9390>

type = <peewee.IntegerField object at 0x24b92d0>

class hanyuu.db.models.**Relay**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *relays* table.

DoesNotExist
alias of RelayDoesNotExist

active = <peewee.IntegerField object at 0x24b9a50>

base_name = <peewee.CharField object at 0x24b9850>

bitrate = <peewee.IntegerField object at 0x24b9910>

country = <peewee.CharField object at 0x24b9ad0>

disabled = <peewee.IntegerField object at 0x24b9b10>

format = <peewee.CharField object at 0x24b9950>

id = <peewee.PrimaryKeyField object at 0x24b9490>

listener_limit = <peewee.IntegerField object at 0x24b9a10>

listeners = <peewee.IntegerField object at 0x24b99d0>

mountpoint = <peewee.CharField object at 0x24b98d0>

owner = <peewee.CharField object at 0x24b9810>

passcode = <peewee.CharField object at 0x24b9a90>

port = <peewee.IntegerField object at 0x24b9890>

priority = <peewee.IntegerField object at 0x24b9990>

subdomain = <peewee.CharField object at 0x24b97d0>

class hanyuu.db.models.**Song**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *esong* table.

DoesNotExist
alias of SongDoesNotExist

faves

classmethod from_meta(*metadata*)
Returns the first match found of metadata

Params unicode metadata A string of metadata.

Returns Song instance.

Raises Song.DoesNotExist if no result was found.

Note: This currently does no pre-fetching of the faves and plays

---

hash = <peewee.CharField object at 0x24b81d0>

hash_link = <peewee.CharField object at 0x24b8290>

id = <peewee.PrimaryKeyField object at 0x24b8110>

length = <peewee.IntegerField object at 0x24b8210>

meta = <peewee.TextField object at 0x24b8250>

plays

classmethod query_from_meta(*metadata*)
Returns the first match found of metadata

Params unicode metadata A string of metadata.

Returns peewee.SelectQuery instance.

Note: This currently does no pre-fetching of the faves and plays

---

queued

class hanyuu.db.models.**Track**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *tracks* table.

DoesNotExist
alias of TrackDoesNotExist

acceptor = <peewee.CharField object at 0x24b8c50>

album = <peewee.CharField object at 0x24b8ad0>

artist = <peewee.CharField object at 0x24b8a50>

filename = <peewee.TextField object at 0x24b8b10>

classmethod from_meta(*metadata*)
Returns the first match found of metadata

Params unicode metadata A string of metadata.

Returns Track instance.

hash = <peewee.CharField object at 0x24b8cd0>

id = <peewee.PrimaryKeyField object at 0x24b8850>

last_editor = <peewee.CharField object at 0x24b8c90>

last_played = <peewee.DateTimeField object at 0x24b8b90>

last_requested = <peewee.DateTimeField object at 0x24b8bd0>

needs_reupload = <peewee.IntegerField object at 0x24b8d90>

priority = <peewee.IntegerField object at 0x24b8d10>

queued

request_count = <peewee.IntegerField object at 0x24b8d50>

search_tags = <peewee.TextField object at 0x24b8b50>

title = <peewee.CharField object at 0x24b8a90>

usable = <peewee.IntegerField object at 0x24b8c10>

class hanyuu.db.models.**User**(**args*, ***kwargs*)
Bases: hanyuu.db.models.Base

Models the legacy *users* table.

DoesNotExist
alias of UserDoesNotExist

dj = <peewee.ForeignKeyField object at 0x24b5650>

id = <peewee.PrimaryKeyField object at 0x24b5350>

name = <peewee.CharField object at 0x24b55d0>

password = <peewee.CharField object at 0x24b5610>

privileges = <peewee.IntegerField object at 0x24b5690>

irc Package

irc Package

commands Module

irclib Module

listener Package

listener Package

requests Package

requests Package

hanyuu.requests.**songdelay**(*val*)
Gives the time delay in seconds for a specific song request count.

Subpackages

servers Package

servers Package

fastcgi Module

status Package

status Package

class hanyuu.status.**Base**
Bases: object

Simple base class that sets the attribute :attr:cache to a :class:memcache.Client ready to be used.

cache

class hanyuu.status.**Site**
Bases: hanyuu.status.Base

Object that encapsulates state of the website.

dj
Returns the current DJ that is live.

Returns a abstractions.users.DJ object.

thread
Returns the current thread URL.

Returns a unicode string or None

class hanyuu.status.**Stream**
Bases: hanyuu.status.Base

Wrapping class around the memcache server and variables relevant to the status of the streaming server.

current
Gets the current song metadata playing on the master server.

Returns a unicode object.

listeners
Returns the total amount of listeners as an integer.

This is the listeners combined from all relay servers.

online
Returns if the master server is online or not.

Returns a boolean type.

peak_listeners

class hanyuu.status.**Streamer**
Bases: hanyuu.status.Base

Object that encapsulates state of the AFK streamer.

requests_enabled
Returns a bool indicating if the AFK streamer accepts requests.

This is False if either Requests got disabled explicitely or the AFK streamer is not streaming at the moment.

hanyuu.status.**memcache_client**()
Returns a pylibmc.Client object.

streamstatus Module

streamer Package

streamer Package

afkstreamer Module

class hanyuu.streamer.afkstreamer.**Streamer**(*attributes*)
Bases: object

Top wrapper of the AFK Streamer. This gives out filenames and metadata to the underlying audio module.

close(*force=False*)
Stop the audio pipeline and disconnects from icecast.

connect(**args*, ***kwargs*)
Deprecated since version 1.2: use start(): instead.

connected
Returns True if the audio modules audio.icecast is currently connected. Else returns False.

shutdown(**args*, ***kwargs*)
Deprecated since version 1.2: use close(): instead.

start()
Starts the audio pipeline and connects to icecast.

supply_song()
Returns a tuple of (filename, metadata) to be played next.

Subpackages

audio Package

audio Package

class hanyuu.streamer.audio.**FileInformation**(*filename*, *metadata=None*)
Bases: object

A class that should be returned from the function passed to Manager for file discovery.

This is to make switching functions easier since the Manager doesn't need to know what format the function returns but only know that it returns a FileInformation instance instead.

class hanyuu.streamer.audio.**Manager**(*source*, *processors=None*, ***options*)
Bases: object

A class that manages the audio pipeline. Each component gets a reference to the processor before it.

Note: This is a very cruel pipeline and has specifics to our needs and is in no way a generic implementation.

Nor does it have proper definitions of what should go out or into a processor.

The Manager expects that all registered processors have at least the following characteristics:

start(): Called when Manager.start() is called. This should initialize required components. The Manager expects that a call to *close* and *start* is close to equal of recreating the whole instance.

close(): Called when Manager.close() is called. This should close down the processor cleanly and if potential long running cleanups are to be done should use the garbage sub package shipped with the **audio** package.

__init__(): Called when the Manager instance is created. This should not start any state dependant parts, these should be done in the *start* method instead.

Gets passed one positional argument that is the previous processor in the chain. Or if the first processor read below.

Gets passed extra keyword arguments if specified in the class attribute *options*. Read more about this attribute below.

The current version expects the first specified processor to take a function as *source* argument. That can be called for the filepath of an audiofile. This first processor is responsible for opening it.

Note: This means the processor doesn't actually need to decode the file but that it is just expected to accept the function as *source*. What it does with the function is not important to the Manager.

The current version expects the last specified processor to have several methods available to be used by the Manager. These are:

status(): A method that is called when status() is called. This should return something of significants to the user.

metadata(): A method that accepts a single *unicode* argument. This is called whenever new metadata is found at the start of the processor chain.

close()
Calls the *close* method on all registered processor instances.

> **Warning:** Exceptions are propagated.

get_source()

Returns unicode A full file path to an audio file.

The value returned from `Manager.source()` is expected to be an `FileInformation` object. But there is one exception to this rule.

When `Manager.source()` returns a different type it will be used as the positional arguments to the `FileInformation` constructor by using the *FileInformation(*returntype)* syntax.

processors = [<class 'hanyuu.streamer.audio.files.FileSource'>, <class 'hanyuu.streamer.audio.encoder.Encoder'>, <cl
A list of processors that are instanced in order and are passed their previous friend as first argument.

start()
Calls the *start* method on all registered processor instances.

This method does nothing if a previous call to `start()` was successful but `close()` was not called in between the two calls.

> **Warning:** Exceptions are propagated.

status()
Calls the *status* method on the last processor in the chain.

If no method was found returns `False` instead.

hanyuu.streamer.audio.**test_config**(*password=None*)

hanyuu.streamer.audio.**test_dir**(*directory=u'/media/F/Music'*, *files=None*)

encoder Module

class hanyuu.streamer.audio.encoder.**Encoder**(*source*, *lame_settings*)
Bases: `object`

An Encoder class that handles the encoder subprocess underneath.

This expects various things from the **source** given.

The source should have the following characteristics:

read(): A function that accepts a single integer argument that is the amount of bytes to return. It should return PCM audio data in a supported format.

sample_rate: The sample rate of the audio data. This should be the full integer of the sample rate (44100 instead of 44.1)

bits_per_sample: The bits per sample of the audio data. This can be 16, 24 and 32 bits.

close()
This calls the `EncoderInstance.close()` method on the `EncoderInstance`.

encoding_settings = None
The settings for encoding to pass to lame as a list.

options = [(u'lame_settings', [u'–cbr', u'-b', u'192', u'–resample', u'44.1'])]

report_close()
This method is called by the EncoderInstance class when it gets closed or an error occurs in the instance. This should handle the case gracefully and even restart the instance if the close was unintentional by the user.

The method registers the EncoderInstance instance for garbage collection by the garbage module.

restart()
This method rather then restart, destroys and then recreates the underlying EncoderInstance instance.

start()
This clears our *alive* flag and starts a new EncoderInstance instance by calling start_instance().

start_instance()
This method is responsible for creating and starting the EncoderInstance class instances.

This creates a new EncoderInstance instance and calls the EncoderInstance.start() method on it.

After the call to 'start' returns the new instance is assigned to instance.

class hanyuu.streamer.audio.encoder.**EncoderInstance**(*encoder_manager*)
Bases: object

Class that represents a subprocessed encoder.

close()

read(*size=4096*, *timeout=10.0*)

run()

start()

switch_source(*new_source*)

write(*data*)

class hanyuu.streamer.audio.encoder.**GarbageInstance**(*item=None*)
Bases: hanyuu.streamer.audio.garbage.Garbage

collect()

hanyuu.streamer.audio.encoder.**LAME_BIN = u'lame'**
The path to the LAME binary. This can be just 'lame' on bash environments.

files Module Module that handles file access and decoding to PCM.

It uses python-audiotools for the majority of the work done.

exception hanyuu.streamer.audio.files.**AudioError**
Bases: exceptions.Exception

Exception raised when an error occurs in this module.

class hanyuu.streamer.audio.files.**AudioFile**(*filename*)
Bases: object

A Simple wrapper around the audiotools library.

This opens the filename given wraps the file in a PCMConverter that turns it into PCM of format 44.1kHz, Stereo, 24-bit depth.

close()
Registers self for garbage collection. This method does not close anything and only registers itself for colleciton.

progress(*current*, *total*)
Dummy progress function

read(*size=4096*, *timeout=0.0*)
Returns at most a string of size *size*.

The *timeout* argument is unused. But kept in for compatibility with other read methods in the *audio* module.

class hanyuu.streamer.audio.files.**FileSource**(*source_function*)
Bases: object

change_source()
Calls the source function and returns the result if not None.

close()

initialize()
Sets the initial source from the source function.

read(*size=4096*, *timeout=10.0*)

skip()

start()
Starts the source

class hanyuu.streamer.audio.files.**GarbageAudioFile**(*item=None*)
Bases: hanyuu.streamer.audio.garbage.Garbage

Garbage class of the AudioFile class

collect()
Tries to close the AudioFile resources when called.

icecast Module

class hanyuu.streamer.audio.icecast.**Icecast**(*source*, *config*)
Bases: object

close()
Closes the libshout object and tries to join the thread if we are not calling this from our own thread.

connect()
Connect the libshout object to the configured server.

connected()
Returns True if the libshout object is currently connected to an icecast server.

connecting_timeout = 5.0
The time to wait when we lose connection by cause of external behaviour.

metadata(*metadata*)

nonblocking(*state*)

options = [('icecast_config', {})]
Options that __init__() should get passed from the pipeline manager when being created. (See hanyuu.streamer.audio.Manager for more information.)

read(*size*, *timeout=None*)

reboot_libshout()
Internal method

Tries to recreate the libshout object.

run()

setup_libshout()
Internal method

Creates a libshout object and puts the configuration to use.

start()
Starts the thread that reads from source and feeds it to icecast.

switch_source(*new_source*)
Tries to change the source without disconnect from icecast.

class hanyuu.streamer.audio.icecast.**IcecastConfig**(*attributes=None*)
Bases: dict

Simple dict subclass that knows how to apply the keys to a libshout object.

setup(*shout*)
Setup 'shout' configuration by setting attributes on the object.

'shout' is a pylibshout.Shout object.

exception hanyuu.streamer.audio.icecast.**IcecastError**
Bases: exceptions.Exception

Subpackages

garbage Package

garbage Package

class hanyuu.streamer.audio.garbage.**Collector**
Bases: object

add(*garbage*)

classmethod add_hook(*hook*)

info()
Returns a list of GarbageInfo objects containing information about current pending garbage.

instance = <hanyuu.streamer.audio.garbage.Collector object at 0x3cda2d0>

run()

class hanyuu.streamer.audio.garbage.**Garbage**(*item=None*)
Bases: object

collect()
Gets called on each collection cycle.

Should return True if the garbage got cleaned up properly, False if it requires another collect in the next cycle.

collector = <hanyuu.streamer.audio.garbage.Collector object at 0x3cda2d0>

class hanyuu.streamer.audio.garbage.**Singleton**(*mcs*, *name*, *bases*, *dict*)
Bases: type

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

h