LMSTools Documentation

Release 0.0.1

elParaguayo

May 17, 2019

Contents:

CHAPTER 1

Introduction

LMSTools is a python library for interacting with a Logitech Media Server.

This code was inspired by the PyLMS library by JingleManSweep and has sought to recreate a lot of the functionality that was provided by that library. The main difference is that the PyLMS library used the server's telnet interface whereas LMSTools uses the JSON interface.

LMSTools also includes additional functionality: an asynchronous callback server and the ability to generate player menus.

CHAPTER 2

Installation

The project is not currently on PyPi so you need to install the library manually.

2.1 Manual installation

Download the zip file from github (it's recommended you stick to the master branch unless you want to test new features).

Unzip the file and copy the LMSTools folder to your project folder or, if you want to use it in multiple projects, copt it to a folder in your python path.

Note: Once the code reaches a stable enough level I will submit to PyPi.

This process can be accelerated by submitting bug reports whenever encountered.

CHAPTER 3

Examples

Please use the links below to see examples of how to use the LMSTools library.

3.1 Starting out

Unsurprisingly, the library is centered around the server. So your first step is to create a server object.

```
from LMSTools import LMSServer

# Define your server address
SERVER_IP = "192.168.0.1"

# create the server object
server = LMSServer(SERVER_IP)
```

At this point, you can test if your connection works by running the Ping method.

```
>>>server.Ping()
True
```

Pretty unexciting though, isn't it?

That's because you know it's not the server that really matters, it's the players. So let's see how they work in the next section: *Controlling/querying your squeezeplayer*.

3.2 Controlling/querying your squeezeplayer

3.2.1 Retrieving players

Once you've got your server, you'll want to get your players next.

It's easy to get the list of the players currently attached to your server:

```
from LMSTools import LMSServer

# Define your server address
SERVER_IP = "192.168.0.1"

# create the server object
server = LMSServer(SERVER_IP)

# get the attached players
players = server.get_players()
```

Each item in 'players' will be a *LMSPlayer* instance and should be easily identifiable by printing the output of the list.

```
>>>server.get_players()
[LMSPlayer: Living Room (40:40:40:40:40:40),
 LMSPlayer: PiRadio (41:41:41:41:41:41),
 LMSPlayer: elParaguayo's Laptop (42:42:42:42:42:42)]
>>>laptop = server.get_players()[2]
>>>Laptop
LMSPlayer: elParaguayo's Laptop (42:42:42:42:42:42)
```

So, now you've got your player, what can you do with it?

3.2.2 Controlling the player

It's easy to do simple manipulation of the playlist.

The player has methods to *play*, *pause* and skip tracks.

```
>>>laptop.play()
>>>laptop.next()
>>>laptop.stop()
>>>
```

3.2.3 Changing volume

Players have a `volume <LMSTools.player.LMSPlayer.volume` property. This can be used to retrieve the current volume level and adjust it. In addition there are *volume_up* and *volume_down* methods.

```
>>># Get the volume level
>>>laptop.volume
75
>>>laptop.volume_down()
>>>laptop.volume
70
>>>laptop.volume_down(10)
>>>laptop.volume
60
>>>laptop.volume = 90
>>>laptop.volume
90
```

3.2.4 Syncing players

You can sync and unsync players easily.

```
>>>livingroom = server.get_players()[0]
>>>livingroom
LMSPlayer: Living Room (40:40:40:40:40:40
>>>laptop.sync(livingroom)
>>>
```

You can confirm which players are synced with your player:

```
>>>laptop.get_synced_players()
[LMSPlayer: Living Room (40:40:40:40:40:40]
>>>
```

If there are multiple sync groups then you can view members by using the *show_players_sync_status* method.

3.2.5 Adding tracks to the playlist

If you have a path to a playable item, these can be added to the playlist directly.

```
>>># You can use spotify urls if the app is installed
>>>laptop.playlist_play("spotify://track:5xYZXIgVAND5sWjN8G0hID")
>>>
```

The *playlist_insert* and *playlist_add* methods can be used to place tracks at different locations in the playlist (i.e. next and last) while *playlist_delete* can be used to remove tracks.

```
>>>laptop.playlist_delete("spotify://track:5xYZXIgVAND5sWjN8G0hID")
>>>
```

3.2.6 Getting metadata

In case you don't know what's actually playing at the moment, you can retrieve metadata about the track (and other items in the playlist).

```
>>>laptop.track_title
u'Go!'
>>>laptop.track_artist
u'Public Service Broadcasting'
>>>laptop.track_album
u'The Race For Space'
```

If you want to query the playlist, there are a number of options open to you. See: *playlist_get_info*, *playlist_get_detail* and *playlist_get_current_detail*.

```
>>>laptop.playlist_get_current_detail()
[{u'album': u'The Race For Space',
  u'artist': u'Public Service Broadcasting',
  u'coverart': u'0',
  u'coverid': u'-186029800',
  u'duration': u'252',
  u'id': u'-186029800',
```

(continues on next page)

(continued from previous page)

```
  u'playlist index': 0,
  u'remote': 1,
  u'title': u'Go!'}]
```

Additional information can be requested by using *tags*.

```
>>>from LMSTools import LMSTags as tags
>>>laptop.playlist_get_current_detail(taglist=[tags.DURATION, tags.CONTENT_TYPE])
[{u'duration': u'252',
  u'id': u'-186029800',
  u'playlist index': 0,
  u'title': u'Go!',
  u'type': u'Ogg Vorbis (Spotify)'}]
```

3.2.7 . . . and more

See the class documentation for *LMSPlayer* for further information on available properties and methods.

3.3 Using the callbackserver

Callbacks can be configured in two different ways:

1) Using decorators
2) Using the 'add_callback' method

Decorators

```
squeeze = LMSCallbackServer()

@squeeze.event(squeeze.VOLUME_CHANGE)
def volume_event(event=None):
    print "Volume event received: {}".format(event)

squeeze.set_server("192.168.0.1")
squeeze.start()
```

If you are using decorators inside a class then this will happen before your class has been initialised so you need to provide the callback server with a reference to the class instance.

```
squeeze = LMSCallbackServer()

class MyClass(object):

    def __init__(self):
        self.squeeze = squeeze
        self.squeeze.set_server("192.168.0.1", parent_class=self)
        self.squeeze.start()

    @squeeze.event(squeeze.VOLUME_CHANGE)
    def volume_event(self, event=None):
        print "Volume event received: {}".format(event)
```

Multiple events can be added with multiple decorators

```
@squeeze.event(squeeze.VOLUME_CHANGE)
@squeeze.event(squeeze.PLAY_PAUSE)
def generic_event(event=None):
    print "Event received: {}".format(event)
```

Or by passing events as a list

```
@squeeze.event([squeeze.VOLUME_CHANGE, squeeze.PLAY_PAUSE])
def generic_event(event=None):
    print "Event received: {}".format(event)
```

Using 'add_callback' method

```
def volume_event(event=None):
    print "Volume event received: {}".format(event)

squeeze = LMSCallbackServer("192.168.0.1")
squeeze.add_callback(squeeze.VOLUME_CHANGE, volume_event)
squeeze.start()
```

3.4 Generating player menus

The *LMSMenuHandler* class allows you to generate squeezeplayer menus on the fly. This will allow you to create your own interfaces in your applications.

Note: This code is a work in progress and may therefore lack some of the functionality that you may encounter on more 'professional' applications.

If there is some functionality that is missing (or the code otherwise works in unexpected ways) then please notify me in the GitHub issues tracker.

3.4.1 Understanding the menu system

Menus are provided by the server as JSON objects. At their most basic, they provide text, icon path and the relevant command to be executed.

This library currently categorises each menu item into one of four types:

- *NextMenuItem*: a menu item which just provides an additional submenu;
- *PlaylistMenuItem*: a menu item which can be played/added to playlist or can provide a subsequent menu showing the tracks in the playlist;
- *AudioMenuItem*: a menu item which can be played/added to playlist; and
- *SearchMenuItem*: a menu item which requires user input before providing results.

The use of these different menu types is set out further below.

3.4.2 Creating a menu handler

The menu handler is currently included as a separate class. As the menus are specific to each player, the menu handler must have information about the player for which the menu is being requested.

```
>>>from LMSTools import LMSServer, LMSMenuHandler
>>>server = LMSServer("192.168.0.1")
>>>laptop = server.get_players()[1]
>>>handler = LMSMenuHandler(laptop)
>>>
```

If you wish to create a menu for a different player then you can change the current player as follows:

```
>>>livingroom = server.get_players()[0]
>>>handler.changePlayer(livingroom)
>>>
```

3.4.3 Generating a menu

To simplify the process of creating a menu, the menu handler has a a built infunction to retrieve the home menu: *getHomeMenu* .

```
>>>home = handler.getHomeMenu()
>>>home
[<LMSTools.menuitems.NextMenuItem at 0x7f049a0de490>,
 <LMSTools.menuitems.NextMenuItem at 0x7f049a0de450>,
 <LMSTools.menuitems.NextMenuItem at 0x7f049a0de9d0>,
 <LMSTools.menuitems.NextMenuItem at 0x7f049a0de510>,
 # etc.
 ]
```

3.4.4 Custom menus

As you can see from the above, the default home menu is very large and may be unwieldy for your own application.

As a result, you may want to define your own menu and have the menu handler process this menu.

```
CUSTOM_MENU = {
    "count": 5,
    "item_loop": [{
        "node": "myMusic",
        "weight": 11,
        "text": "Artists",
        "actions": {
            "go": {
                "cmd": ["browselibrary", "items"],
                "params": {
                    "menu": 1,
                    "mode": "artists",
                    "role_id": "ALBUMARTIST,ARTIST,BAND,COMPOSER,CONDUCTOR,TRACKARTIST
↪"
                }
            }
        },
        "icon": "html/images/artists.png"
    }, {
        "node": "myMusic",
        "text": "Albums",
        "actions": {
```

(continues on next page)

(continued from previous page)

```
            "go": {
                "cmd": ["browselibrary", "items"],
                "params": {
                    "menu": 1,
                    "mode": "albums"
                }
            }
        },
        "id": "myMusicAlbums",
        "icon": "html/images/albums.png"
    }, {
        "node": "myMusic",
        "text": "Playlists",
        "icon": "html/images/playlists.png",
        "actions": {
            "go": {
                "cmd": ["browselibrary", "items"],
                "params": {
                    "menu": 1,
                    "mode": "playlists"
                }
            }
        }
    }, {
         "node": "myMusic",
         "text": "Search",
         "icon": "html/images/search.png",
         "actions": {
            "go": {
                "cmd": ["browselibrary", "items"],
                "params": {
                    "menu": 1,
                    "mode": "search"
                }
            }
        }
    }, {
        "node": "home",
        "window": {
            "titleStyle": "album",
            "icon-id": "plugins/MyApps/html/images/icon.png"
        },
        "text": "My Apps",
        "actions": {
            "go": {
                "player": 0,
                "cmd": ["myapps", "items"],
                "params": {
                    "menu": "myapps"
                }
            }
        }
    }]
    }

menu = LMSMenuHandler(player)
results = menu.getCustomMenu(CUSTOM_MENU)
```

(continues on next page)

(continued from previous page)

```
for item in results:
    print item.text, item.cmd
```

should output the following

```
Artists ['browselibrary', 'items', 0, 1000, 'menu:1', 'mode:artists', 'role_
↪id:ALBUMARTIST,ARTIST,BAND,COMPOSER,CONDUCTOR,TRACKARTIST']
Albums ['browselibrary', 'items', 0, 1000, 'menu:1', 'mode:albums']
Playlists ['browselibrary', 'items', 0, 1000, 'menu:1', 'mode:playlists']
Search ['browselibrary', 'items', 0, 1000, 'menu:1', 'mode:search']
My Apps ['myapps', 'items', 0, 1000, 'menu:myapps']
```

3.4.5 Navigating the menu

Next Menu items

NextMenuItems' purpose is to take the user to another menu. The object therefore provides the necessary command required to generate the next menu:

```
>>>next = home[0]
>>>menu = handler.getMenu(next.go())
>>>
```

Playlist Menu items

In addition to providing a submenu (i.e. a list of the tracks in the playlist) a playlist meny item can be played/added to the queue.

```
>>># Assume this is a playlist menu item!
>>>playlist = home[0]
>>>playlist.play()
>>>
```

See the *PlaylistMenuItem* class documentation for more information.

Audio Menu items

These behave the same as Playlist Menu items (with the exception that they don't provide a submenu of playable tracks).

Search Menu items

These items require user input to deliver tailored responses.

```
>>># Assume this is a search menu item!
>>>searchitem = home[0]
>>>cmd = searchitem.search("My search term")
>>>results = handler.getMenu(cmd)
>>>
```

CHAPTER 4

Module documentation

4.1 LMSServer

Simple python class definitions for interacting with Logitech Media Server. This code uses the JSON interface.

exception LMSTools.server.**LMSConnectionError**

class LMSTools.server.**LMSServer**(*host='localhost'*, *port=9000*)

Parameters

- **host** (*str*) – address of LMS server (default "localhost")
- **port** (*int*) – port for the web interface (default 9000)

Class for Logitech Media Server. Provides access via JSON interface. As the class uses the JSON interface, no active connections are maintained.

get_player_count()

Return type int

Returns number of connected players

```
>>>server.get_player_count()
3
```

get_players()

Return type list

Returns list of LMSPlayer instances

Return a list of currently connected Squeezeplayers.

```
>>>server.get_players()
[LMSPlayer: Living Room (40:40:40:40:40:40),
 LMSPlayer: PiRadio (41:41:41:41:41:41),
 LMSPlayer: elParaguayo's Laptop (42:42:42:42:42:42)]
```

get_sync_groups()

Return type list

Returns list of syncgroups. Each group is a list of references of the members.

```
>>>server.get_sync_groups()
[[u'40:40:40:40:40:40', u'41:41:41:41:41:41']]
```

ping()

Return type bool

Returns True if server is alive, False if server is unreachable

Method to test if server is active.

```
>>>server.ping()
True
```

request(*player='-'*, *params=None*)

Parameters

- **player** (*(str)*) – MAC address of a connected player. Alternatively, "-" can be used for server level requests.
- **params** (*(str, list)*) – Request command

rescan(*mode='fast'*)

Parameters **mode** (*str*) – Mode can be 'fast' for update changes on library, 'full' for complete library scan and 'playlists' for playlists scan only

Trigger rescan of the media library.

rescanprogress

Attr rescanprogress current rescan progress

show_players_sync_status()

Return type dict

Returns dictionary (see attributes below)

Attr group_count (int) Number of sync groups

Attr player_count (int) Number of connected players

Attr players (list) List of players (see below)

Player object (dict)

Attr name Name of player

Attr ref Player reference

Attr sync_index Index of sync group (-1 if not synced)

```
>>>server.show_players_sync_status()
{'group_count': 1,
 'player_count': 3,
 'players': [{'name': u'Living Room',
              'ref': u'40:40:40:40:40:40',
              'sync_index': 0},
```

(continues on next page)

(continued from previous page)

```
                {'name': u'PiRadio',
                'ref': u'41:41:41:41:41:41',
                'sync_index': 0},
                {'name': u"elParaguayo's Laptop",
                'ref': u'42:42:42:42:42:42',
                'sync_index': -1}]}
```

sync(*master*, *slave*)

Parameters

- **master** (*(ref)*) – Reference of the player to which you wish to sync another player
- **slave** (*(ref)*) – Reference of the player which you wish to sync to the master

Sync squeezeplayers.

version

Attr version Version number of server Software

```
>>>server.version
u'7.9.0'
```

4.2 LMSPlayer

class LMSTools.player.**LMSPlayer**(*ref*, *server*)

The LMSPlayer class represents an individual squeeze player connected to your Logitech Media Server.

Instances of this class are generated from the LMSServer object and it is not expected that you would create an instance directly. However, it is posible to create instances directly:

```
server = LMSServer("192.168.0.1")

# Get player instance with MAC address of player
player = LMSPlayer("12:34:56:78:90:AB", server)

# Get player based on index of player on server
player = LMSPlayer.from_index(0, server)
```

Upon intialisation, basic information about the player is retrieved from the server:

```
>>>player = LMSPlayer("12:34:56:78:90:AB", server)
>>>player.name
u'Living Room'
>>>player.model
u'squeezelite'
```

forward(*seconds=10*)

Parameters **seconds** (*int, float*) – number of seconds to jump forwards in current track.

Jump forward in current track. Number of seconds will be converted to integer.

classmethod from_index(*index*, *server*)

Create an instance of LMSPlayer when the MAC address of the player is unknown.

This class method uses the index of the player (as registered on the server) to identify the player.

Return type *LMSPlayer*

Returns Instance of squeezeplayer

get_synced_players(*refs_only=False*)
Retrieve list of players synced to current player.

Parameters **refs_only** (*bool*) – whether the method should return list of MAC references or list of LMSPlayer instances.

Return type list

mode

Return type str, unicode

Returns curent mode (e.g. "play", "pause")

model

Return type str, unicode

Returns model name of the current player.

mute()
Mute player

muted
Muting

Getter retrieve current muting status

Return type bool

Returns True if muted, False if not.

Setter set muting status (True = muted)

name
Player name.

Getter retrieve name of player

Return type unicode, str

Returns name of player

Setter update name of player on server

```
>>>p.name
u"elParaguayo's Laptop"
>>>p.name = "New name"
>>>p.name
'New name'
```

next()
Play next item in playlist

parse_request(*command*, *key*)

Parameters

- **command** (*str, list*) – command to be sent to server
- **key** (*str*) – key to retrieve desired info from JSON response

Returns value from JSON response

Send the request and extract the info from the JSON response.

This is the same as player.request(command).get(key)

pause()
Pause the player. This does not unpause the player if already paused.

percentage_elapsed(*upper=100*)

Parameters **upper** (*float, int*) – (optional) scale - returned value is between 0 and upper (default 100)

Return type float

Returns current percentage elapsed

```
>>>player.percentage_elapsed()
29.784033576552005
>>>p.percentage_elapsed(upper=1)
0.31738374576051237
```

play()
Start playing the current item

playlist_add(*item*)
Add item to playlist

Parameters **item** (*str*) – link to playable item

playlist_clear()
Clear the entire playlist. Will also stop the player.

playlist_delete(*item*)
Delete item

Parameters **item** (*str*) – link to playable item

playlist_erase(*index*)
Remove item from playlist by index

Parameters **index** – index of item to delete

playlist_get_current_detail(*amount=None*, *taglist=None*)

Parameters

- **amount** (*int*) – number of tracks to query
- **taglist** (*list*) – list of tags (NEED LINK)

Return type list

Returns server result

If amount is None, all remaining tracks will be displayed.

If not taglist is provided, the default list is: [tags.ARTIST, tags.COVERID, tags.DURATION, tags.COVERART, tags.ARTWORK_URL, tags.ALBUM, tags.REMOTE, tags.ARTWORK_TRACK_ID]

```
>>>player.playlist_get_current_detail(amount=1)
[{u'album': u'Jake Bugg',
  u'artist': u'Jake Bugg',
  u'artwork_url': u'https://i.scdn.co/image/
↪6ba50b26867613b100281669ff1a917c5a020534',
  u'coverart': u'0',
```

(continues on next page)

(continued from previous page)

```
  u'coverid': u'-161090728',
  u'duration': u'144',
  u'id': u'-161090728',
  u'playlist index': 7,
  u'remote': 1,
  u'title': u'Lightning Bolt'}]
>>>player.playlist_get_current_detail(amount=1, taglist=[tags.DURATION])
[{u'duration': u'144',
  u'id': u'-161090728',
  u'playlist index': 7,
  u'title': u'Lightning Bolt'}]
```

playlist_get_detail(*start=None*, *amount=None*, *taglist=None*)

Parameters

- **start** (*int*) – playlist index of first track to query
- **amount** (*int*) – number of tracks to query
- **taglist** (*list*) – list of tags (NEED LINK)

Return type list

Returns server result

If start is None, results will start with the first track in the playlist.

If amount is None, all playlist tracks will be returned.

If not taglist is provided, the default list is: [tags.ARTIST, tags.COVERID, tags.DURATION, tags.COVERART, tags.ARTWORK_URL, tags.ALBUM, tags.REMOTE, tags.ARTWORK_TRACK_ID]

```
>>>player.playlist_get_detail(start=1, amount=1, taglist=[tags.URL])
[{u'id': u'-137990288',
 u'playlist index': 1,
 u'title': u"Mardy Bum by Arctic Monkeys from Whatever People Say I Am, That
↪'s What I'm Not",
 u'url': u'spotify://track:2fyIS6GXMgUcSv4oejx63f'}]
```

playlist_get_info(*taglist=None*, *start=None*, *amount=None*)

Parameters

- **start** (*int*) – playlist index of first track to query
- **amount** (*int*) – number of tracks to query
- **taglist** (*list*) – list of tags (NEED LINK)

Return type list

Returns server result

If start is None, results will start with the first track in the playlist.

If amount is None, all playlist tracks will be returned.

Unlike playlist_get_detail, no default taglist is provided.

```
>>>player.playlist_get_info(start=1, amount=1)
[{u'id': u'-137990288',
```

(continues on next page)

(continued from previous page)

```
 u'playlist index': 1,
 u'title': u'Mardy Bum'}]
```

playlist_insert(*item*)
Insert item into playlist (after current track)

Parameters **item** (*str*) – link to playable item

playlist_move(*from_index*, *to_index*)
Move items in playlist

Parameters

- **from_index** (*int*) – index of item to move
- **to_index** (*int*) – new playlist position

playlist_play(*item*)
Play item

Parameters **item** (*str*) – link to playable item

playlist_play_index(*index*)

Parameters **index** (*int*) – index of playlist track to play (zero-based index)

playlist_position

Return type int

Returns position of current track in playlist

prev()
Play previous item in playlist

request(*command*)

Parameters **command** (*str, list*) – command to be sent to server

Return type dict

Returns JSON response received from server

Send the request to the server.

rewind(*seconds=10*)

Parameters **seconds** (*int, float*) – number of seconds to jump backwards in current track.

Jump backwards in current track. Number of seconds will be converted to integer.

seek_to(*seconds*)

Parameters **seconds** (*int, float*) – position (in seconds) that player should seek to

Move player to specified position in current playlist item

stop()
Stop the player

sync(*player=None*, *ref=None*, *index=None*, *master=True*)
Synchronise squeezeplayers

Parameters

- **player** (LMSPlayer) – Instance of player

- **ref** (*str*) – MAC address of player
- **index** (*int*) – server index of squeezeplayer
- **master** (*bool*) – whether current player should be the master player in sync group

Raises LMSPlayerError

You must provide one of player, ref or index otherwise an exception will be raised. If master is set to True then you must provide either player or ref.

time_elapsed

Return type float

Returns elapsed time in seconds. Returns 0.0 if an exception is encountered.

time_remaining

Return type float

Returns remaining time in seconds. Returns 0.0 if an exception is encountered.

toggle()
Play/Pause Toggle

track_album

Return type unicode, str

Returns name of album for current playlist item

```
>>>player.track_album
u'Kiasmos'
```

track_artist

Return type unicode, str

Returns name of artist for current playlist item

```
>>>player.track_artist
u'Kiasmos'
```

track_count

Return type int

Returns number of tracks in playlist

track_duration

Return type float

Returns duration of track in seconds

```
>>>player.track_duration
384.809
```

track_elapsed_and_duration

Return type tuple (float, float)

Returns tuple of elapsed time and track duration

```
>>>player.track_elapsed_and_duration
(4.86446976280212, 384.809)
```

track_title

Return type unicode, str

Returns name of track for current playlist item

```
>>>player.track_artist
u'Lit'
```

unmute()
Unmute player

unpause()
Unpause the player.

unsync()
Remove player from syncgroup.

update()
Retrieve some basic info about the player.

Retrieves the name, model and ip attributes. This method is called on initialisation.

volume
Volume information

Getter Get current volume

Return type int

Returns current volume

Setter change volume

```
>>>player.volume
95
>>>player.volume = 50
```

Min: 0, Max: 100

volume_down(*interval=5*)
Decrease volume

Parameters **interval** (*int*) – amount to decrease volume (default 5)

volume_up(*interval=5*)
Increase volume

Parameters **interval** (*int*) – amount to increase volume (default 5)

wifi_signal_strength

Return type int

Returns Wifi signal strength

4.3 Callback Server

An asynchronous client that listens to messages broadcast by the server.

The client also accepts callback functions.

The client subclasses python threading so methods are built-in to the class object.

Callbacks can be configured in two different ways:

1) Using decorators
2) Using the 'add_callback' method

Decorators

```
squeeze = LMSCallbackServer()

@squeeze.event(squeeze.VOLUME_CHANGE)
def volume_event(event=None):
    print "Volume event received: {}".format(event)

squeeze.set_server("192.168.0.1")
squeeze.start()
```

If you are using decorators inside a class then this will happen before your class has been initialised so you need to provide the callback server with a reference to the class instance.

```
squeeze = LMSCallbackServer()

class MyClass(object):

    def __init__(self):
        self.squeeze = squeeze
        self.squeeze.set_server("192.168.0.1", parent_class=self)
        self.squeeze.start()

    @squeeze.event(squeeze.VOLUME_CHANGE)
    def volume_event(self, event=None):
        print "Volume event received: {}".format(event)
```

Multiple events can be added with multiple decorators

```
@squeeze.event(squeeze.VOLUME_CHANGE)
@squeeze.event(squeeze.PLAY_PAUSE)
def generic_event(event=None):
    print "Event received: {}".format(event)
```

Or by passing events as a list

```
@squeeze.event([squeeze.VOLUME_CHANGE, squeeze.PLAY_PAUSE])
def generic_event(event=None):
    print "Event received: {}".format(event)
```

Using 'add_callback' method

```
def volume_event(event=None):
    print "Volume event received: {}".format(event)

squeeze = LMSCallbackServer("192.168.0.1")
squeeze.add_callback(squeeze.VOLUME_CHANGE, volume_event)
squeeze.start()
```

exception LMSTools.callbackserver.**CallbackServerError**

class LMSTools.callbackserver.**LMSCallbackServer**(*hostname=None*, *port=9090*, *username=''*, *password=''*)

Parameters

- **hostname** (*str*) – (optional) ip address/name of the server (excluding "http://" prefix)
- **port** (*int*) – (optional) port on which the telent interface is running (default 9090)
- **username** (*str*) – (optional) username for access on telnet port
- **password** (*str*) – (optional) password for access on telnet port

If the class is initialised without the hostname parameter then the "set_server" method must be called before starting the server otherwise a CallbackServerError will be raised.

Events

The following events are currently define in the class.

Const MIXER_ALL Captures all mixer events

Const VOLUME_CHANGE Captures volume events

Const PLAYLIST_ALL Captures all playlist events

Const PLAY_PAUSE Captures play/pause events

Const PLAY Captures play event

Const PAUSE Captures pause event

Const PLAYLIST_OPEN Captures playlist open event

Const PLAYLIST_CHANGE_TRACK Captures track changes

Const PLAYLIST_LOAD_TRACKS Captures loadtracks event

Const PLAYLIST_ADD_TRACKS Captures addtracks event

Const PLAYLIST_LOADED Captures "playlist load_done" event

Const PLAYLIST_REMOVE Captures "playlist delete" event

Const PLAYLIST_CLEAR Captures playlist clear event

Const PLAYLIST_CHANGED Captures PLAYLIST_LOAD_TRACKS, PLAYLIST_LOADED, PLAYLIST_ADD_TRACKS, PLAYLIST_REMOVE, PLAYLIST_CLEAR

Const CLIENT_ALL Captures all client events

Const CLIENT_NEW Captures new client events

Const CLIENT_DISCONNECT Captures client disconnect events

Const CLIENT_RECONNECT Captures client reconnect events

Const CLIENT_FORGET Captures client forget events

Const SYNC Captures sync events

Const SERVER_ERROR Custom event for server errors

Const SERVER_CONNECT Custom event for server connection

add_callback(*event*, *callback*)

Define a callback.

Parameters

- **event** (*event*) – Event type

- **callback** (*function/method*) – Reference to the function/method to be called if matching event is received. The function/method must accept one parmeter which is the event string.

remove_callback(*event*)
Remove a callback.

Parameters **event** (*event*) – Event type

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

set_server(*hostname*, *port=9090*, *username=''*, *password=''*, *parent_class=None*)

Parameters

- **hostname** (*str*) – (required) ip address/name of the server (excluding "http://" prefix)
- **port** (*int*) – (optional) port on which the telent interface is running (default 9090)
- **username** (*str*) – (optional) username for access on telnet port
- **password** (*str*) – (optional) password for access on telnet port
- **parent_class** (*object*) – (optional) reference to a class instance. Required where decorators have been used on class methods prior to initialising the class.

Provide details of the server if not provided when the class is initialised (e.g. if you are using decorators to define callbacks).

stop()
Stop the callack server thread.

4.4 Squeezeplayer Menus

exception LMSTools.menu.**LMSMenuException**
Simple exception class for handling errors in the LMSMenuHandler.

class LMSTools.menu.**LMSMenuHandler**(*player=None*)

Parameters **player** (LMSPlayer) – instance of LMSPlayer.

Class for generating squeezeplayer menu for individual players.

This code is a work in progress and currently has limited functionality.

Menus can be requested via the getHomeMenu or getCustomMenu methods. Subsequent menus are generated by getting the menu command from previous menus and passing it to the getMenu method.

If no player is set when the handler is initiated then it must be set before requesting menus.

changePlayer(*player*)

Parameters **player** (LMSPlayer) – instance of LMSPlayer.

Change the player for which the menu is being created.

While this may have little relevance on retrieiving menus, it is important if you wish to manipulate the playlist or other player specific items directly from the menu.

dump(*menu*, *filename*)

Parameters

- **menu** (*dict*) – raw json menu
- **filename** (*str*) – name of file to save menu

Save the supplied menu to file (useful for debugging purposes).

getCustomMenu(*raw*)

Parameters **raw** (*dict*) – custom menu format (see docs for example)

Return type list

Returns list of menu items

Generate menu items from a custom menu.

This can be useful if you want to create a tailored menu rather than use the full default menu generated by the server.

getHomeMenu()

Return type list

Returns list of menu items

Generate menu items from default menu.

getMenu(*menucmd*)

Parameters **menucmd** (*str, list*) – command to request next menu from server

Return type list

Returns list of menu items

Generate menu from the supplied menu command.

class LMSTools.menuitems.**AudioMenuItem**(*player=None*, *menuitem=None*, *base=None*)

Audio menu item. Basically the same as a playlist.

add()

Add the selected item to your playlist.

cmd_add

Return type str

Returns command string to add selected item to playlist

cmd_play

Return type str

Returns command string to play selected item

cmd_play_next

Return type str

Returns command string to play selected item after currently playing item

go()

Return type list

Returns command list for submenu

Go to submenu i.e. list of tracks in playlist.

play()
Play the selected item.

play_next()
Play the selected item after the currently playing item.

show_items_cmd

Return type str

Returns command string to show submenu items

class LMSTools.menuitems.**NextMenuItem**(*player=None*, *menuitem=None*, *base=None*)
Menu item which has no other purpose than to create a new submenu.

cmd

Return type str

Returns command string for next menu

Get command string for submenu.

class LMSTools.menuitems.**PlaylistMenuItem**(*player=None*, *menuitem=None*, *base=None*)
A playlist menu item is one that can be played directly from this link but can also provide a submenu of all the tracks in the playlist.

add()
Add the selected item to your playlist.

cmd_add

Return type str

Returns command string to add selected item to playlist

cmd_play

Return type str

Returns command string to play selected item

cmd_play_next

Return type str

Returns command string to play selected item after currently playing item

go()

Return type list

Returns command list for submenu

Go to submenu i.e. list of tracks in playlist.

play()
Play the selected item.

play_next()
Play the selected item after the currently playing item.

show_items_cmd

Return type str

Returns command string to show submenu items

class LMSTools.menuitems.**SearchMenuItem**(*player=None*, *menuitem=None*, *base=None*)
Menu item where a search term is required.

cmd_search

Return type str

Returns raw command string

You will need to replace __TAGGEDINPUT__ with your search term before building a menu with this command.

search(*query*)

Parameters **query** (*str*) – search terms

Return type list

Returns command to generate search results

4.5 Server command tags

class LMSTools.tags.**LMSTags**

Const ARTIST Artist name.

Const ALBUM_ID Album ID. Only if known.

Const ALBUM_REPLAY_GAIN Replay gain of the album (in dB), if any

Const ALBUM Album name. Only if known.

Const ARTIST_ID Artist ID.

Const ARTIST_ROLE_IDS For each role as defined above, the list of ids.

Const ARTIST_ROLE a comma separated list of names.

Const ARTWORK_TRACK_ID Identifier of the album track used by the server to display the album's artwork. Not listed if artwork is not available for this album.

Const ARTWORK_URL A full URL to remote artwork. Only available for certain plugins such as Pandora and Rhapsody.

Const BITRATE Song bitrate. Only if known.

Const BPM Beats per minute. Only if known.

Const BUTTONS A hash with button definitions. Only available for certain plugins such as Pandora.

Const COMMENT Song comments, if any.

Const COMPILATION 1 if the album this track belongs to is a compilation

Const CONTENT_TYPE Content type. Only if known.

Const COVERART 1 if coverart is available for this song. Not listed otherwise.

Const COVERID coverid to use when constructing an artwork URL, such as /music/$coverid/cover.jpg

Const DISC_COUNT Number of discs. Only if known.

Const DISC Disc number. Only if known.

Const DURATION Song duration in seconds.

Const FILESIZE Song file length in bytes. Only if known.

Const GENRE_ID_LIST Genre IDs, separated by commas (only useful if the server is set to handle multiple items in tags).

Const GENRE_ID Genre ID. Only if known.

Const GENRE_LIST Genre names, separated by commas (only useful if the server is set to handle multiple items in tags).

Const GENRE Genre name. Only if known.

Const INFO_LINK A custom link to use for trackinfo. Only available for certain plugins such as Pandora.

Const LYRICS Lyrics. Only if known.

Const MODIFICATION_TIME Date and time song file was last changed.

Const MUSICMAGIC_MIXABLE 1 if track is mixable, otherwise 0.

Const RATING Song rating, if known and greater than 0.

Const REMOTE_TITLE Title of the internet radio station.

Const REMOTE If 1, this is a remote track.

Const REPLAY_GAIN Replay gain (in dB), if any

Const SAMPLERATE Song sample rate (in KHz)

Const SAMPLESIZE Song sample size (in bits)

Const TAG_VERSION Version of tag information in song file. Only if known.

Const TRACK_NUMBER Track number. Only if known.

Const URL Song file url.

Const YEAR Song year. Only if known.

4.6 Artwork

Note: There is limited documentation for this class as it is expected that the functionality will be added to the LMSPlayer class.

class LMSTools.artworkresolver.**LMSArtworkResolver**(*host='localhost'*, *port=9000*)

Class object to help provide an easy way of obtaining a URL to a playlist item.

The class is capable of working out the appropriate path depending on whether the file is remote or local.

Parameters

- **host** (*str*) – address of the server
- **port** (*int*) – webport of the server (default 9000)

getURL(*track*, *size=(500, 500)*)

Method for generating link to artwork for the selected track.

Parameters

- **track** (*dict*) – a dict object which must contain the "remote", "coverid" and "coverart" tags as returned by the server.
- **size** (*tuple*) – optional parameter which can be used when creating links for local images. Default (500, 500).

CHAPTER 5

Contributing

5.1 Enhancements

Requests should be submitted on the Issues tracker on Github.

Please submit any pull requests to the development branch. Requests to master will be rejected.

5.2 Bugs

Bugs should be logged on the Issues tracker .

CHAPTER 6

License

As PyLMS was licensed under GPL v2, this library has used the same license.

Attention: LMSTools: A python library for interacting with a Logitech Media Server

Copyright (C) 2017 elParaguayo

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to:

Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

M

N

P

R

S

T

U

V

W