plexdevices Documentation

Release 0.4.1

Cory Parsons

May 02, 2016

Contents

A Python module for working with plex devices.

```
>>> import plexdevices
>>> s = plexdevices.create_session(user=username, password=password)
>>> s.refresh_devices()
>>> s.servers
[<Device:Server1 - Plex Media Server>, <Device:Server2 - Plex Media Server>]
>>> on_deck = s.servers[0].media_container('/library/onDeck')
>>> on_deck
<plexdevices.media.MediaContainer object at 0x03BD32B0>
>>> on_deck.children
[<Movie:Movie>, <Episode:Episode>, <Episode:Episode>, <Episode:Episode>]
>>> on_deck.children[0].media[0].parts[0].resolve_url()
http://server/file.mp4?X-Plex-Token=XXXXXXXXXXX
```

CHAPTER 1

API Documentation

1.1 Main Interface

plexdevices.**create_session**(*user=None*, *password=None*, *token=None*)

Create a *Session*.

Parameters

- **user** – (optional) Plex.tv username.
- **password** – (optional) Plex.tv password.
- **token** – (optional) an *X-Plex-Token*.

Returns *Session* object.

Return type *plexdevices.session.Session*

Plex.TV Usage:

```
>>> import plexdevices
>>> s = plexdevices.create_session(username, password)
>>> s.refresh_devices()
>>> on_deck = s.servers[0].media_container('/library/onDeck')
>>> on_deck.children[0].resolve_url()
http://server/file.mp4?X-Plex-Token=XXXXXXXXXXX
```

Manual Usage:

```
>>> import plexdevices
>>> s = plexdevices.create_session()
>>> s.manual_add_server('192.168.1.1', 32400)
>>> s.manual_add_server('192.168.1.2', 32400, token='w1zWSdJXzdeQEnpgdmLxB')
>>> s.servers
[<Device:A - Plex Media Server>, <Device:B - Plex Media Server>]
```

note: manually connecting to a server requires either an *X-Plex-Token* (see Plex support article) or an ip address added to the `List of networks that are allowed without auth` PMS Network setting.

plexdevices.**create_remote**(*player*, *name*, *port=8000*, *callback=None*)

Create a *Remote*.

Parameters

- **player** – the *Player* to create the remote for.
- **name** – the name of the remote.

- **port** – (optional) HTTP port to allow the Player to communicate to the remote.
- **callback** – (optional) a function that will be called with a single argument when the player *POSTs* data. (when using the subscribe method).

Returns *Remote* object.

Return type *plexdevices.remote.Remote*

Setup:

```
>>> import plexdevices
>>> s = plexdevices.create_session(username, password)
>>> s.refresh_devices()
>>> s.players
[<Device:My-PC - Plex Media Player>]
>>> player = s.players[0]
```

Usage (subscribe method):

```
>>> r = plexdevices.create_remote(player, 'myremote')
>>> r.timeline_subscribe()
>>> r.down()
>>> r.select()
>>> r.timeline_unsubscribe()
```

Usage (poll method):

```
>>> r = plexdevices.create_remote(player, 'myremote')
>>> timeline_data = r.timeline_poll()
>>> timeline_data
{
    'video': {'type': 'video', 'time': '0', 'seekRange': '0-0'},
    'photo': {'type': 'photo', 'time': '0', 'seekRange': '0-0',
              'controllable': 'playPause,stop,skipPrevious,skipNext'}
    'music': {'type': 'music', 'time': '0', 'seekRange': '0-0',
              'controllable': 'playPause,stop,skipPrevious,skipNext,seekTo'},
    'location': 'navigation',
    'commandID': '0'
}
>>> r.down()
>>> r.select()
```

1.2 Sessions

class plexdevices.session.**Session**

A Plex session. You can use pickle to save and load existing session objects.

login(*password*)

Retrieve the token for the session user from https://plex.tv/users/sign_in.json.

manual_add_server(*address*, *port=32400*, *protocol='http'*, *token=''*)

Add a *Server* to the session.

Parameters

- **address** – address to the server. e.g. 127.0.0.1.
- **token** – (optional) the X-Plex-Token to use when accessing this server.

players = None
List of *Player*'s accessible by the current user.

refresh_devices()
Retrieve the devices for the current user from `https://plex.tv/api/resources`

refresh_users()
Retrieve the Plex Home users from `https://plex.tv/api/home/users`.

servers = None
List of *Server*'s accessible by the current user.

switch_user(*user_id*, *pin=None*)
Switch the current user to the given user id, and refresh the available devices.

Parameters

- **user_id** – the *id* of the user. As given from `https://plex.tv/api/home/users`.
- **pin** – (optional) the 4-digit PIN code of the user.

users = None
List of Plex Home users.

1.3 Devices

1.3.1 Device

class plexdevices.device.**Device**

access_token

client_identifier
Unique identifier string.

created_at

device

headers

https_required

Type bool

last_seen_at

name

owned

Type bool

platform
Operating system of the device.

platform_version
Operating system version.

presence

Type bool

product
Plex product name. e.g. `Plex Media Server`

product_version
Version of the Plex product. e.g. `0.9.16.4.1911-ee6e505`

provides

Type list

public_address_matches

Type bool

request(*endpoint*, *method='GET'*, *data=None*, *params=None*, *headers={}*, *raw=False*, *allow_redirects=True*)
Make an HTTP request to the device.

Parameters

- **endpoint** – location on server. e.g. `/library/onDeck`.
- **method** – (optional) request function. Defaults to `GET`.
- **data** – (optional) data to send with the request. Defaults to `None`.
- **params** – (optional) params to include in the URL. Defaults to `None`.
- **headers** – (optional) additional headers. Defaults to `{}`.
- **raw** – (optional) return raw data. Defaults to `False`.
- **allow_redirects** – (optional) follow 302 redirects. Defaults to `True`.

Returns (HTTP status code, data)

Return type Tuple (int, str)

synced

Type bool

1.3.2 Server

class plexdevices.device.**Server**
Bases: *plexdevices.device.Device*

A *Device* which provides a server.

container(*endpoint*, *size=None*, *page=None*, *params=None*, *usejson=True*)

Parameters

- **endpoint** – destination on the server. e.g. `/library/onDeck`.
- **size** – (optional) the max number of items to retrieve.
- **page** – (optional) the page number for paging large containers.
- **params** – (optional) Dictionary of parameters to be added to the url in the request.

Returns a Dictionary representing a Plex Media Container.

Return type Dictionary

hub(*endpoint*, *size=None*, *page=None*, *params=None*)
added in 0.4.0

Parameters

- **endpoint** – destination on the hubs api. e.g. `/hubs/onDeck`.
- **params** – (optional) Dictionary of parameters to be added to the url in the request.

Return type *HubsContainer*

image(*endpoint*, *w=None*, *h=None*)
If w and h are set, the server will transcode the image to the given size.

Parameters

- **endpoint** – location of the image. This can also be a full URL of an image not on the server (for easy channel support).
- **w** – (optional) width to transcode.
- **h** – (optional) height to transcode.

Returns Raw data of an image.

media_container(*endpoint*, *size=None*, *page=None*, *params=None*, *usejson=True*)

Parameters

- **endpoint** – destination on the server. e.g. `/library/onDeck`.
- **size** – (optional) the max number of items to retrieve.
- **page** – (optional) the page number for paging large containers.
- **params** – (optional) Dictionary of parameters to be added to the url in the request.

Returns a *MediaContainer* representing a Plex Media Container.

Return type *MediaContainer*

1.3.3 Player

class plexdevices.device.**Player**
Bases: *plexdevices.device.Device*

A *Device* which provides a player.

1.4 Containers

1.4.1 Media Container

class plexdevices.media.**MediaContainer**
An object representing a Plex MediaContainer.

children = None
List of items in the container.

data = None
Dictionary of the MediaContainer's values.

server = None
The *Server* which this container was retrieved from.

1.4.2 Play Queues

class plexdevices.media.**PlayQueue**
Bases: *plexdevices.media.MediaContainer*

An object representing a Plex PlayQueue. A PlayQueue is a playlist that is maintained by the server.

When you want to play a Media item, use the `create()` method. Then during playback, use `timeline_update()` to let the server know about the players state.

add_item(*item*, *player_headers*)
Add a *MediaDirectory* or *MediaItems* to the PlayQueue.

static create(*item*, *player_headers*)
Create a PlayQueue on a server and return a PlayQueue object.

Parameters

- **item** – the *Media* to be the initial item added to the PlayQueue.
- **player_headers** – Dictionary of headers identifying the player using the PlayQueue. Must include X-Plex-Client-Identifier and X-Plex-Device-Name.

Returns *PlayQueue* object

Return type plexdevices.PlayQueue

get_next()
Select and return the next *MediaItem* in the PlayQueue, or *None*.

get_prev()
Select and return the previous *MediaItem* in the PlayQueue, or *None*.

remove_item(*item*)
Remove a *MediaItem* from the PlayQueue.

selected_item
The selected *MediaItem*

timeline_update(*item*, *time*, *headers*, *state='playing'*)
Update the timeline. This should be done frequently during playback. The server will update the view offset of the item and handle the watched/unwatched state.

Parameters

- **item** – the play queue item.
- **time** – the current playback time in ms.
- **headers** – the players headers. `X-Plex-Client-Identifier`, `X-Plex-Device-Name`.
- **state** – playing, stopped, paused

update()
Update the *PlayQueue*'s data from its server.

1.4.3 Hubs

added in 0.4.0

These are for the Plex Media Server api at `/hubs`.

class `plexdevices.hubs.`**HubsContainer**
Bases: *plexdevices.media.MediaContainer*

allow_sync

children = None
List of items in the container. `BaseObject`, *Hub*

data = None
Dictionary of the HubsContainer's values.

is_library

server = None
The `Server` which this container was retrieved from.

size

class `plexdevices.hubs.`**Hub**
A Hub is a container inside a *HubsContainer* which groups together multiple items.

children = None
List of `BaseObject`'s in the Hub.

container = None
The *HubsContainer* which holds this item.

hub_identifier

hub_key

key

more

size

title

type

1.5 Items in Containers

Deciding what to do with the items in a container should be done by checking the type with `isinstance()`.

```
def item_clicked(item):
    if isinstance(item, (plexdevices.media.Movie, plexdevices.media.Episode)):
        video_player.play(item.resolve_url())
    elif isinstance(item, plexdevices.media.Directory):
        next_container = server.media_container(item.key)
```

1.5.1 Directories

class plexdevices.media.**Directory**
Bases: plexdevices.media.BaseObject

a directory that is used for navigation.

key
The endpoint on the server which this object points to. This can be relative or absolute.

title

class plexdevices.media.**InputDirectory**
Bases: *plexdevices.media.Directory*

A special *Directory* where you should get a string from the user and send it to the key as a *query* parameter.

```
if isinstance(item, InputDirectory):
    user_input = get_string()
    next_container = server.media_container(item.key, params={'query': user_input})
```

class plexdevices.media.**PreferencesDirectory**
Bases: *plexdevices.media.Directory*

A special *Directory* used in channels for channel preferences.

1.5.2 Media Directories

MediaDirectory

class plexdevices.media.**MediaDirectory**
Bases: *plexdevices.media.Directory*, plexdevices.media.Metadata

A directory that holds MediaItems. These directories have metadata, can be added to a *PlayQueue*, and can be marked watched/unwatched. A *MediaContainer* with its key will contain all the *MediaItems*.

added_at
Unix timestamp.

allow_sync

Type bool

art
Art key.

key
The endpoint on the server which this object points to. This can be relative or absolute.

mark_unwatched()
Mark this item as unwatched on its server.

mark_watched()
Mark this item as watched on its server.

summary
Description of the item.

thumb
Either the key to the thumb, or None. Use image() to get the image data.

title

updated_at
Unix timestamp.

PhotoAlbum

class plexdevices.media.**PhotoAlbum**
Bases: *plexdevices.media.MediaDirectory*

MediaDirectory with extra metadata for a Photo Album.

Album

class plexdevices.media.**Album**
Bases: *plexdevices.media.MediaDirectory*

MediaDirectory with extra metadata for a music Album.

genres

Type list

originally_available_at
Release date.

parent_key
Artist key.

parent_title
Artist name.

year

Artist

class plexdevices.media.**Artist**
Bases: *plexdevices.media.MediaDirectory*

MediaDirectory with extra metadata for a music Artist.

country

Type list

genres

Type list

Season

class plexdevices.media.**Season**
Bases: *plexdevices.media.MediaDirectory*

MediaDirectory with extra metadata for a TV Season.

count
Number of episodes.

parent_key
Show key.

parent_summary
Show summary.

parent_thumb
Show thumb.

parent_title
Show name.

unwatched_count

watched_count

Show

class plexdevices.media.**Show**
Bases: *plexdevices.media.MediaDirectory*

MediaDirectory with extra metadata for a TV Show.

banner

child_count
Number of seasons.

count
Number of episodes.

duration
Length of one episode in ms.

originally_available_at
Premiere date.

studio

unwatched_count

watched_count

1.5.3 Media Items

MediaItem

class plexdevices.media.**MediaItem**
Bases: plexdevices.media.BaseObject, plexdevices.media.Metadata

An object representing a piece of Media.

added_at
Unix timestamp.

allow_sync

Type bool

art
Art key.

in_progress
True if the item is in progress.

key
The endpoint on the server which this object points to. This can be relative or absolute.

last_viewed_at
not always available.

mark_unwatched()
Mark this item as unwatched on its server.

mark_watched()
Mark this item as watched on its server.

media = None
List of *Media* objects that hold information about the file.

resolve_url()
Return the url of the first part regardless of how many there are.

summary
Description of the item.

thumb
Either the key to the thumb, or `None`. Use `image()` to get the image data.

title

updated_at
Unix timestamp.

view_offset
The resume position in ms.

watched
`True` if the item is watched.

year

Media

class plexdevices.media.**Media**
A Media object represents a single copy of a *MediaItem*. In most cases, a *MediaItem* will have a single Media object. If the server has a 480p and a 1080p copy of a movie, there will be two Media objects.

aspect_ratio
Aspect ratio as floating point number. e.g. `1.78` for 16:9.

audio_channels
Number of audio channels.

audio_codec
Audio codec as a string.

bitrate

container
Container file format. e.g. `mkv`.

duration
Duration in ms.

height
Vertical resolution.

id

parts = None
List of *Part* objects which references the actual files. Typically there is only one part.

video_codec
Video codec as a string.

video_frame_rate
Framerate as a string. e.g. `24p`, `NTSC`.

video_profile
`main`, `high`, ...

video_resolution
Vertical resolution as an integer. e.g. `1080`

width
Horizontal resolution.

Parts

class `plexdevices.media.`**Part**
A part represents an actual file.

container
Container file format. e.g. `mkv`.

duration
Duration in ms.

file
The path to the file on the server. use the key to actually access it.

id

key
The key to playing this item. It either points to the file on the server, or to a function on the server that will resolve it to a playable url.

resolve_key`()`
Resolve the key into a url which can be given to a media player.

size
Size of the file in bytes.

video_profile
`main`, `high`, ...

Episode

class `plexdevices.media.`**Episode**
Bases: *plexdevices.media.MediaItem*

MediaItem with extra metadata for a TV Show Episode.

duration
Duration in ms.

grandparent_key
Key of the show.

grandparent_title
Title of the show.

index
Episode number.

originally_available_at
Air date.

parent_index
Season number.

parent_key
Key of the season.

Movie

class plexdevices.media.**Movie**
Bases: *plexdevices.media.MediaItem*

MediaItem with extra metadata for a Movie.

duration
Duration in ms.

originally_available_at

Type datetime.date

rating
Movie rating as a floating point number.

studio

tagline

Track

class plexdevices.media.**Track**
Bases: *plexdevices.media.MediaItem*

MediaItem with extra metadata for a Music Track.

duration
Duration in ms.

grandparent_key
Artist key.

grandparent_thumb

grandparent_title
Artist name.

index
Track number.

parent_key
Album key.

parent_thumb

parent_title
Album name.

Photo

class plexdevices.media.**Photo**
Bases: *plexdevices.media.MediaItem*

MediaItem with extra metadata for a Photo.

grandparent_key
Key to the folder which holds the photoalbum.

originally_available_at

parent_key
Key to the album which holds this photo.

1.6 Remote

class plexdevices.remote.**Remote**
A remote control for a Plex device.

identifier = None
The unique identifier string for this device.

name = None

player = None

port = None

headers
Dictionary of the remote's `X-Plex-Client-Identifier` and `X-Plex-Device-Name`.

command(*command*, *params=None*)
Send a command to the player with optional parameters.

timeline()
Returns the latest timeline that was POSTed from the player.

timeline_subscribe()
Subscribe to the timeline.

timeline_unsubscribe()
Unsubscribe from the timeline.

timeline_poll()
This is an alternative to the subscribe command for controllers that cannot use persistent connections to receive updates from the player.

mirror(*plex_object*, ***kwargs*)
Send the player to the preplay screen of the given `BaseObject`.

play_media(*media_object*)
Make the player play the given *MediaItem*.

up()
Navigation: up

down()
Navigation: down

left()
Navigation: left

right()
Navigation: right

select()
Navigation: select

back()
Navigation: back

home()
Navigation: home

music()
Navigation: music

Navigate to the player's music playback view, if music is currently playing.

pause(*media_type=None*)
Playback control: pause

Parameters **media_type** – (optional) `music, photo, video` in case there are multiple things happening.

play(*media_type=None*)
Playback control: play

skip_next(*media_type=None*)
Playback control: skip next

skip_previous(*media_type=None*)
Playback control: skip previous

stop(*media_type=None*)
Playback control: stop

seek(*media_type*, *offset*)
Playback control: seek to

Parameters **offset** – absolute position in milliseconds.

skip(*media_type*, *key*)
Playback control: skip to item with matching key

step_back(*media_type*)
Playback control: step back

step_forward(*media_type*)
Playback control: step forward

volume(*media_type*, *level*)
Playback control: set volume.

Parameters

- **media_type** – mandatory. `music, photo, video`.
- **level** – volume level [0-100].

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

p

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P