aio-androidtv Documentation

Release 0.0.4

Jeff Irion

May 18, 2020

CONTENTS

CHAPTER

ONE

AIO_ANDROIDTV

1.1 aio_androidtv package

1.1.1 Submodules

aio_androidtv.adb_manager module

Classes to manage ADB connections.

- *ADBPython* utilizes a Python implementation of the ADB protocol.

class aio_androidtv.adb_manager.**ADBPython**(*host*, *port*, *adbkey=''*)
Bases: object

A manager for ADB connections that uses a Python implementation of the ADB protocol.

Parameters

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)
- **adbkey** (*str*) – The path to the adbkey file for ADB authentication

property available
Check whether the ADB connection is intact.

Returns Whether or not the ADB connection is intact

Return type bool

async close()
Close the ADB socket connection.

async connect(*always_log_errors=True*, *auth_timeout_s=0.1*)
Connect to an Android TV / Fire TV device.

Parameters

- **always_log_errors** (*bool*) – If True, errors will always be logged; otherwise, errors will only be logged on the first failed reconnect attempt
- **auth_timeout_s** (*float*) – Authentication timeout (in seconds)

Returns Whether or not the connection was successfully established and the device is available

Return type bool

async pull(*local_path*, *device_path*)
Pull a file from the device using the Python ADB implementation.

Parameters

- **local_path** (*str*) – The path where the file will be saved
- **device_path** (*str*) – The file on the device that will be pulled

async push(*local_path*, *device_path*)
Push a file to the device using the Python ADB implementation.

Parameters

- **local_path** (*str*) – The file that will be pushed to the device
- **device_path** (*str*) – The path where the file will be saved on the device

async screencap()
Take a screenshot using the Python ADB implementation.

Returns The screencap as a binary .png image

Return type bytes

async shell(*cmd*)
Send an ADB command using the Python ADB implementation.

Parameters **cmd** (*str*) – The ADB command to be sent

Returns The response from the device, if there is a response

Return type str, None

aio_androidtv.adb_manager.**DEFAULT_TIMEOUT = 3.0**
Default timeout for acquiring the async lock that protects ADB commands

aio_androidtv.adb_manager.**_acquire**(*lock*, *timeout=3.0*)
Handle acquisition and release of an asyncio.Lock object with a timeout.

Parameters

- **lock** (*asyncio.Lock*) – The lock that we will try to acquire
- **timeout** (*float*) – The timeout in seconds

Yields **acquired** (*bool*) – Whether or not the lock was acquired

Raises ***LockNotAcquiredException*** – Raised if the lock was not acquired

aio_androidtv.androidtv module

Communicate with an Android TV device via ADB over a network.

ADB Debugging must be enabled.

class aio_androidtv.androidtv.**AndroidTV**(*host*, *port=5555*, *adbkey=''*, *state_detection_rules=None*)
Bases: *aio_androidtv.basetv.BaseTV*

Representation of an Android TV device.

Parameters

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)
- **adbkey** (*str*) – The path to the adbkey file for ADB authentication

- **state_detection_rules** (*dict, None*) – A dictionary of rules for determining the state (see *BaseTV*)

DEVICE_CLASS = 'androidtv'

async get_properties(*get_running_apps=True*, *lazy=False*)

Get the properties needed for Home Assistant updates.

This will send one of the following ADB commands:

- `aio_androidtv.constants.CMD_ANDROIDTV_PROPERTIES_LAZY`
- `aio_androidtv.constants.CMD_ANDROIDTV_PROPERTIES_NOT_LAZY`

Parameters

- **get_running_apps** (*bool*) – Whether or not to get the *running_apps()* property
- **lazy** (*bool*) – Whether or not to continue retrieving properties if the device is off or the screensaver is running

Returns

- **screen_on** (*bool, None*) – Whether or not the device is on, or `None` if it was not determined
- **awake** (*bool, None*) – Whether or not the device is awake (screensaver is not running), or `None` if it was not determined
- **audio_state** (*str, None*) – The audio state, as determined from "dumpsys audio", or `None` if it was not determined
- **wake_lock_size** (*int, None*) – The size of the current wake lock, or `None` if it was not determined
- **current_app** (*str, None*) – The current app property, or `None` if it was not determined
- **media_session_state** (*int, None*) – The state from the output of `dumpsys media_session`, or `None` if it was not determined
- **audio_output_device** (*str, None*) – The current audio playback device, or `None` if it was not determined
- **is_volume_muted** (*bool, None*) – Whether or not the volume is muted, or `None` if it was not determined
- **volume** (*int, None*) – The absolute volume level, or `None` if it was not determined
- **running_apps** (*list, None*) – A list of the running apps, or `None` if it was not determined

async get_properties_dict(*get_running_apps=True*, *lazy=True*)

Get the properties needed for Home Assistant updates and return them as a dictionary.

Parameters

- **get_running_apps** (*bool*) – Whether or not to get the *running_apps()* property
- **lazy** (*bool*) – Whether or not to continue retrieving properties if the device is off or the screensaver is running

Returns A dictionary with keys `'screen_on'`, `'awake'`, `'wake_lock_size'`, `'current_app'`, `'media_session_state'`, `'audio_state'`, `'audio_output_device'`, `'is_volume_muted'`, `'volume'`, and `'running_apps'`

Return type dict

async running_apps()
Return a list of running user applications.

Returns A list of the running apps

Return type list

async turn_off()
Send `POWER` action if the device is not off.

async turn_on()
Send `POWER` action if the device is off.

async update(*get_running_apps=True*)
Get the info needed for a Home Assistant update.

Parameters **get_running_apps** (*bool*) – Whether or not to get the *running_apps()* property

Returns

- **state** (*str*) – The state of the device
- **current_app** (*str*) – The current running app
- **running_apps** (*list*) – A list of the running apps if `get_running_apps` is True, otherwise the list `[current_app]`
- **audio_output_device** (*str*) – The current audio playback device
- **is_volume_muted** (*bool*) – Whether or not the volume is muted
- **volume_level** (*float*) – The volume level (between 0 and 1)

aio_androidtv.basetv module

Communicate with an Android TV or Amazon Fire TV device via ADB over a network.

ADB Debugging must be enabled.

class aio_androidtv.basetv.**BaseTV**(*host*, *port=5555*, *adbkey=''*, *state_detection_rules=None*)
Bases: object

Base class for representing an Android TV / Fire TV device.

The `state_detection_rules` parameter is of the format:

```
state_detection_rules = {'com.amazon.tv.launcher': ['idle'],
                         'com.netflix.ninja': ['media_session_state'],
                         'com.ellation.vrv': ['audio_state'],
                         'com.hulu.plus': [{'playing': {'wake_lock_size' : 4}},
                                           {'paused': {'wake_lock_size': 2}}],
                         'com.plexapp.android': [{'paused': {'media_session_state
↪': 3, 'wake_lock_size': 1}},
                                                 {'playing': {'media_session_state
↪': 3}},
                                                 'idle']}
```

The keys are app IDs, and the values are lists of rules that are evaluated in order.

VALID_STATES

```
VALID_STATES = ('idle', 'off', 'playing', 'paused', 'standby')
```

Valid rules:

- `'idle'`, `'playing'`, `'paused'`, `'standby'`, or `'off'` = always report the specified state when this app is open
- `'media_session_state'` = try to use the *media_session_state()* property to determine the state
- `'audio_state'` = try to use the *audio_state()* property to determine the state
- `{'<VALID_STATE>': {'<PROPERTY1>': VALUE1, '<PROPERTY2>': VALUE2, ...}}` = check if each of the properties is equal to the specified value, and if so return the state
  - The valid properties are `'media_session_state'`, `'audio_state'`, and `'wake_lock_size'`

Parameters

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)
- **adbkey** (*str*) – The path to the `adbkey` file for ADB authentication
- **state_detection_rules** (*dict, None*) – A dictionary of rules for determining the state (see above)

static _audio_output_device(*stream_music*)

Get the current audio playback device from the `STREAM_MUSIC` block from `adb shell dumpsys audio`.

Parameters **stream_music** (*str, None*) – The `STREAM_MUSIC` block from `adb shell dumpsys audio`

Returns The current audio playback device, or `None` if it could not be determined

Return type str, None

static _audio_state(*audio_state_response*)

Parse the *audio_state()* property from the output of the command *aio_androidtv.constants.CMD_AUDIO_STATE*.

Parameters **audio_state_response** (*str, None*) – The output of the command *aio_androidtv.constants.CMD_AUDIO_STATE*

Returns The audio state, or `None` if it could not be determined

Return type str, None

static _conditions_are_true(*conditions*, *media_session_state=None*, *wake_lock_size=None*, *audio_state=None*)

Check whether the conditions in `conditions` are true.

Parameters

- **conditions** (*dict*) – A dictionary of conditions to be checked (see the `state_detection_rules` parameter in *BaseTV*)
- **media_session_state** (*int, None*) – The *media_session_state()* property
- **wake_lock_size** (*int, None*) – The *wake_lock_size()* property

- **audio_state** (*str, None*) – The *audio_state()* property

Returns Whether or not all the conditions in `conditions` are true

Return type bool

static _current_app(*current_app_response*)

Get the current app from the output of the command *aio_androidtv.constants.CMD_CURRENT_APP*.

Parameters **current_app_response** (*str, None*) – The output from the ADB command *aio_androidtv.constants.CMD_CURRENT_APP*

Returns The current app, or `None` if it could not be determined

Return type str, None

static _current_app_media_session_state(*media_session_state_response*)

Get the current app and the media session state properties from the output of *aio_androidtv.constants.CMD_MEDIA_SESSION_STATE_FULL*.

Parameters **media_session_state_response** (*str, None*) – The output of *aio_androidtv.constants.CMD_MEDIA_SESSION_STATE_FULL*

Returns

- **current_app** (*str, None*) – The current app, or `None` if it could not be determined
- **media_session_state** (*int, None*) – The state from the output of the ADB shell command, or `None` if it could not be determined

_custom_state_detection(*current_app=None*, *media_session_state=None*, *wake_lock_size=None*, *audio_state=None*)

Use the rules in `self._state_detection_rules` to determine the state.

Parameters

- **current_app** (*str, None*) – The *current_app()* property
- **media_session_state** (*int, None*) – The *media_session_state()* property
- **wake_lock_size** (*int, None*) – The *wake_lock_size()* property
- **audio_state** (*str, None*) – The *audio_state()* property

Returns The state, if it could be determined using the rules in `self._state_detection_rules`; otherwise, `None`

Return type str, None

async _get_stream_music(*stream_music_raw=None*)

Get the `STREAM_MUSIC` block from the output of the command *aio_androidtv.constants.CMD_STREAM_MUSIC*.

Parameters **stream_music_raw** (*str, None*) – The output of the command *aio_androidtv.constants.CMD_STREAM_MUSIC*

Returns The `STREAM_MUSIC` block from the output of *aio_androidtv.constants.CMD_STREAM_MUSIC*, or `None` if it could not be determined

Return type str, None

static _is_volume_muted(*stream_music*)

Determine whether or not the volume is muted from the `STREAM_MUSIC` block from `adb shell dumpsys audio`.

Parameters **stream_music** (*str, None*) – The `STREAM_MUSIC` block from `adb shell dumpsys audio`

Returns Whether or not the volume is muted, or `None` if it could not be determined

Return type bool, None

async _key(*key*)
Send a key event to device.

Parameters **key** (*str, int*) – The Key constant

static _media_session_state(*media_session_state_response*, *current_app*)
Get the state from the output of *aio_androidtv.constants.CMD_MEDIA_SESSION_STATE*.

Parameters

- **media_session_state_response** (*str, None*) – The output of *aio_androidtv.constants.CMD_MEDIA_SESSION_STATE*
- **current_app** (*str, None*) – The current app, or `None` if it could not be determined

Returns The state from the output of the ADB shell command, or `None` if it could not be determined

Return type int, None

static _parse_getevent_line(*line*)
Parse a line of the output received in `learn_sendevent`.

Parameters **line** (*str*) – A line of output from `learn_sendevent`

Returns The properly formatted `sendevent` command

Return type str

static _running_apps(*running_apps_response*)
Get the running apps from the output of `aio_androidtv.constants.CMD_RUNNING_APPS`.

Parameters **running_apps_response** (*str, None*) – The output of `aio_androidtv.constants.CMD_RUNNING_APPS`

Returns A list of the running apps, or `None` if it could not be determined

Return type list, None

async _send_intent(*pkg*, *intent*, *count=1*)
Send an intent to the device.

Parameters

- **pkg** (*str*) – The command that will be sent is `monkey -p <pkg> -c <intent> <count>; echo $?`
- **intent** (*str*) – The command that will be sent is `monkey -p <pkg> -c <intent> <count>; echo $?`
- **count** (*int, str*) – The command that will be sent is `monkey -p <pkg> -c <intent> <count>; echo $?`

Returns A dictionary with keys `'output'` and `'retcode'`, if they could be determined; otherwise, an empty dictionary

Return type dict

_volume(*stream_music*, *audio_output_device*)
Get the absolute volume level from the `STREAM_MUSIC` block from `adb shell dumpsys audio`.

Parameters

- **stream_music** (*str, None*) – The `STREAM_MUSIC` block from `adb shell dumpsys audio`
- **audio_output_device** (*str, None*) – The current audio playback device

Returns The absolute volume level, or `None` if it could not be determined

Return type int, None

_volume_level(*volume*)
Get the relative volume level from the absolute volume level.

Parameters **volume** (*int, None*) – The absolute volume level

Returns The volume level (between 0 and 1), or `None` if it could not be determined

Return type float, None

static _wake_lock_size(*wake_lock_size_response*)
Get the size of the current wake lock from the output of *aio_androidtv.constants.CMD_WAKE_LOCK_SIZE*.

Parameters **wake_lock_size_response** (*str, None*) – The output of *aio_androidtv.constants.CMD_WAKE_LOCK_SIZE*

Returns The size of the current wake lock, or `None` if it could not be determined

Return type int, None

async adb_close()
Close the ADB connection.

async adb_connect(*always_log_errors=True*, *auth_timeout_s=0.1*)
Connect to an Android TV / Fire TV device.

Parameters

- **always_log_errors** (*bool*) – If True, errors will always be logged; otherwise, errors will only be logged on the first failed reconnect attempt
- **auth_timeout_s** (*float*) – Authentication timeout (in seconds)

Returns Whether or not the connection was successfully established and the device is available

Return type bool

async adb_pull(*local_path*, *device_path*)
Pull a file from the device.

This calls *aio_androidtv.adb_manager.ADBPython.pull()*.

Parameters

- **local_path** (*str*) – The path where the file will be saved
- **device_path** (*str*) – The file on the device that will be pulled

async adb_push(*local_path*, *device_path*)
Push a file to the device.

This calls *aio_androidtv.adb_manager.ADBPython.push()*.

Parameters

- **local_path** (*str*) – The file that will be pushed to the device

- **device_path** (*str*) – The path where the file will be saved on the device

async adb_screencap()
Take a screencap.

This calls *aio_androidtv.adb_manager.ADBPython.screencap()*.

Returns The screencap as a binary .png image

Return type bytes

async adb_shell(*cmd*)
Send an ADB command.

This calls *aio_androidtv.adb_manager.ADBPython.shell()*.

Parameters **cmd** (*str*) – The ADB command to be sent

Returns The response from the device, if there is a response

Return type str, None

async audio_output_device()
Get the current audio playback device.

Returns The current audio playback device, or `None` if it could not be determined

Return type str, None

async audio_state()
Check if audio is playing, paused, or idle.

Returns The audio state, as determined from the ADB shell command *aio_androidtv.constants.CMD_AUDIO_STATE*, or `None` if it could not be determined

Return type str, None

property available
Whether the ADB connection is intact.

Returns Whether or not the ADB connection is intact

Return type bool

async awake()
Check if the device is awake (screensaver is not running).

Returns Whether or not the device is awake (screensaver is not running)

Return type bool

async back()
Send back action.

async current_app()
Return the current app.

Returns The ID of the current app, or `None` if it could not be determined

Return type str, None

async down()
Send down action.

async enter()
Send enter action.

async get_device_properties()
Return a dictionary of device properties.

Returns props – A dictionary with keys `'wifimac'`, `'ethmac'`, `'serialno'`, `'manufacturer'`, `'model'`, and `'sw_version'`

Return type dict

async home()
Send home action.

async is_volume_muted()
Whether or not the volume is muted.

Returns Whether or not the volume is muted, or `None` if it could not be determined

Return type bool, None

async key_0()
Send 0 keypress.

async key_1()
Send 1 keypress.

async key_2()
Send 2 keypress.

async key_3()
Send 3 keypress.

async key_4()
Send 4 keypress.

async key_5()
Send 5 keypress.

async key_6()
Send 6 keypress.

async key_7()
Send 7 keypress.

async key_8()
Send 8 keypress.

async key_9()
Send 9 keypress.

async key_a()
Send a keypress.

async key_b()
Send b keypress.

async key_c()
Send c keypress.

async key_d()
Send d keypress.

async key_e()
Send e keypress.

async key_f()
Send f keypress.

async key_g()
Send g keypress.

async key_h()
Send h keypress.

async key_i()
Send i keypress.

async key_j()
Send j keypress.

async key_k()
Send k keypress.

async key_l()
Send l keypress.

async key_m()
Send m keypress.

async key_n()
Send n keypress.

async key_o()
Send o keypress.

async key_p()
Send p keypress.

async key_q()
Send q keypress.

async key_r()
Send r keypress.

async key_s()
Send s keypress.

async key_t()
Send t keypress.

async key_u()
Send u keypress.

async key_v()
Send v keypress.

async key_w()
Send w keypress.

async key_x()
Send x keypress.

async key_y()
Send y keypress.

async key_z()
Send z keypress.

async launch_app(*app*)
Launch an app.

Parameters **app** (*str*) – The ID of the app that will be launched

async learn_sendevent(*timeout_s=8*)
Capture an event (e.g., a button press) via `getevent` and convert it into `sendevent` commands.

For more info, see:

- http://ktnr74.blogspot.com/2013/06/emulating-touchscreen-interaction-with.html?m=1
- https://qatesttech.wordpress.com/2012/06/21/turning-the-output-from-getevent-into-something-something-that-can-be

Parameters **timeout_s** (*int*) – The timeout in seconds to wait for events

Returns The events converted to `sendevent` commands

Return type str

async left()
Send left action.

async media_next_track()
Send media next action (results in fast-forward).

async media_pause()
Send media pause action.

async media_play()
Send media play action.

async media_play_pause()
Send media play/pause action.

async media_previous_track()
Send media previous action (results in rewind).

async media_session_state()
Get the state from the output of `dumpsys media_session`.

Returns The state from the output of the ADB shell command `dumpsys media_session`, or `None` if it could not be determined

Return type int, None

async media_stop()
Send media stop action.

async menu()
Send menu action.

async mute_volume()
Mute the volume.

async power()
Send power action.

async right()
Send right action.

async screen_on()
Check if the screen is on.

Returns Whether or not the device is on

Return type bool

async set_volume_level(*volume_level*)
Set the volume to the desired level.

Parameters **volume_level** (*float*) – The new volume level (between 0 and 1)

Returns The new volume level (between 0 and 1), or `None` if `self.max_volume` could not be determined

Return type float, None

async sleep()
Send sleep action.

async space()
Send space keypress.

async start_intent(*uri*)
Start an intent on the device.

Parameters **uri** (*str*) – The intent that will be sent is `am start -a android.intent.action.VIEW -d <uri>`

async stop_app(*app*)
Stop an app.

Parameters **app** (*str*) – The ID of the app that will be stopped

Returns The output of the `am force-stop` ADB shell command, or `None` if the device is unavailable

Return type str, None

async up()
Send up action.

async volume()
Get the absolute volume level.

Returns The absolute volume level, or `None` if it could not be determined

Return type int, None

async volume_down(*current_volume_level=None*)
Send volume down action.

Parameters **current_volume_level** (*float, None*) – The current volume level (between 0 and 1); if it is not provided, it will be determined

Returns The new volume level (between 0 and 1), or `None` if `self.max_volume` could not be determined

Return type float, None

async volume_level()
Get the relative volume level.

Returns The volume level (between 0 and 1), or `None` if it could not be determined

Return type float, None

async volume_up(*current_volume_level=None*)
Send volume up action.

Parameters **current_volume_level** (*float, None*) – The current volume level (between 0 and 1); if it is not provided, it will be determined

Returns The new volume level (between 0 and 1), or `None` if `self.max_volume` could not be determined

Return type float, None

async wake_lock_size()
Get the size of the current wake lock.

Returns The size of the current wake lock, or `None` if it could not be determined

Return type int, None

aio_androidtv.basetv.**state_detection_rules_validator**(*rules*, *exc=<class 'KeyError'>*)
Validate the rules (i.e., the `state_detection_rules` value) for a given app ID (i.e., a key in `state_detection_rules`).

For each `rule` in `rules`, this function checks that:

- `rule` is a string or a dictionary
- If `rule` is a string:
  - Check that `rule` is in *VALID_STATES* or *VALID_STATE_PROPERTIES*
- If `rule` is a dictionary:
  - Check that each key is in *VALID_STATES*
  - Check that each value is a dictionary
    - Check that each key is in *VALID_PROPERTIES*
    - Check that each value is of the right type, according to *VALID_PROPERTIES_TYPES*

See *BaseTV* for more info about the `state_detection_rules` parameter.

Parameters

- **rules** (*list*) – A list of the rules that will be used to determine the state
- **exc** (*Exception*) – The exception that will be raised if a rule is invalid

Returns **rules** – The provided list of rules

Return type list

aio_androidtv.constants module

Constants used in the *BaseTV*, *AndroidTV*, and *FireTV* classes.

Links

- ADB key event codes
- MediaSession PlaybackState property

aio_androidtv.constants.**CMD_ANDROIDTV_PROPERTIES_LAZY_NO_RUNNING_APPS = "(dumpsys power | g**
Get the properties for an *AndroidTV* device (`lazy=True, get_running_apps=False`); see *aio_androidtv.androidtv.AndroidTV.get_properties()*

aio_androidtv.constants.**CMD_ANDROIDTV_PROPERTIES_LAZY_RUNNING_APPS = "(dumpsys power | grep**
Get the properties for an *AndroidTV* device (`lazy=True, get_running_apps=True`); see *aio_androidtv.androidtv.AndroidTV.get_properties()*

aio_androidtv.constants.**CMD_ANDROIDTV_PROPERTIES_NOT_LAZY_NO_RUNNING_APPS = "(dumpsys powe**
Get the properties for an *AndroidTV* device (`lazy=False, get_running_apps=False`); see *aio_androidtv.androidtv.AndroidTV.get_properties()*

aio_androidtv.constants.**CMD_ANDROIDTV_PROPERTIES_NOT_LAZY_RUNNING_APPS = "(dumpsys power |**
Get the properties for an *AndroidTV* device (`lazy=False, get_running_apps=True`); see *aio_androidtv.androidtv.AndroidTV.get_properties()*

aio_androidtv.constants.**CMD_ANDROIDTV_RUNNING_APPS = 'ps -A | grep u0_a'**
Get the running apps for an Android TV device

aio_androidtv.constants.**CMD_AUDIO_STATE = "dumpsys audio | grep paused | grep -qv 'Buffer (**
Get the audio state

aio_androidtv.constants.**CMD_AWAKE = 'dumpsys power | grep mWakefulness | grep -q Awake'**
Determine whether the device is awake

aio_androidtv.constants.**CMD_CURRENT_APP = 'CURRENT_APP=$(dumpsys window windows | grep mCu**
Get the current app

aio_androidtv.constants.**CMD_FIRETV_PROPERTIES_LAZY_NO_RUNNING_APPS = "(dumpsys power | grep**
Get the properties for a *FireTV* device (`lazy=True, get_running_apps=False`); see *aio_androidtv.firetv.FireTV.get_properties()*

aio_androidtv.constants.**CMD_FIRETV_PROPERTIES_LAZY_RUNNING_APPS = "(dumpsys power | grep 'I**
Get the properties for a *FireTV* device (`lazy=True, get_running_apps=True`); see *aio_androidtv.firetv.FireTV.get_properties()*

aio_androidtv.constants.**CMD_FIRETV_PROPERTIES_NOT_LAZY_NO_RUNNING_APPS = "(dumpsys power |**
Get the properties for a *FireTV* device (`lazy=False, get_running_apps=False`); see *aio_androidtv.firetv.FireTV.get_properties()*

aio_androidtv.constants.**CMD_FIRETV_PROPERTIES_NOT_LAZY_RUNNING_APPS = "(dumpsys power | gre**
Get the properties for a *FireTV* device (`lazy=False, get_running_apps=True`); see *aio_androidtv.firetv.FireTV.get_properties()*

aio_androidtv.constants.**CMD_FIRETV_RUNNING_APPS = 'ps | grep u0_a'**
Get the running apps for a Fire TV device

aio_androidtv.constants.**CMD_LAUNCH_APP = "CURRENT_APP=$(dumpsys window windows | grep mCurr**
Launch an app if it is not already the current app

aio_androidtv.constants.**CMD_MEDIA_SESSION_STATE = "dumpsys media_session | grep -A 100 'Ses**
Get the state from `dumpsys media_session`; this assumes that the variable `CURRENT_APP` has been defined

aio_androidtv.constants.**CMD_MEDIA_SESSION_STATE_FULL = "CURRENT_APP=$(dumpsys window windo**
Determine the current app and get the state from `dumpsys media_session`

aio_androidtv.constants.**CMD_SCREEN_ON = "(dumpsys power | grep 'Display Power' | grep -q 's**
Determine if the device is on

aio_androidtv.constants.**CMD_STREAM_MUSIC = "dumpsys audio | grep '\\- STREAM_MUSIC:' -A 12'**
Get the "STREAM_MUSIC" block from `dumpsys audio`

aio_androidtv.constants.**CMD_WAKE_LOCK_SIZE = "dumpsys power | grep Locks | grep 'size='"**
Get the wake lock size

aio_androidtv.constants.**DEFAULT_AUTH_TIMEOUT_S = 0.1**
Default authentication timeout (in s) for `adb_shell.tcp_handle.TcpHandle.connect()`

aio_androidtv.constants.**MEDIA_SESSION_STATES = {0: None, 1: 'stopped', 2: 'paused', 3:**
States for the *media_session_state* property

aio_androidtv.constants.**VALID_PROPERTIES = ('audio_state', 'media_session_state', 'wake_loc**
Properties that can be checked for custom state detection (used by *state_detection_rules_validator()*)

aio_androidtv.constants.**VALID_PROPERTIES_TYPES = {'audio_state': <class 'str'>, 'media_ses**
The required type for each entry in *VALID_PROPERTIES* (used by *state_detection_rules_validator()*)

aio_androidtv.constants.**VALID_STATES = ('idle', 'off', 'playing', 'paused', 'standby')**
States that are valid (used by *state_detection_rules_validator()*)

aio_androidtv.constants.**VALID_STATE_PROPERTIES = ('audio_state', 'media_session_state')**
Properties that can be used to determine the current state (used by *state_detection_rules_validator()*)

aio_androidtv.exceptions module

Exceptions for use throughout the code.

exception aio_androidtv.exceptions.**LockNotAcquiredException**
Bases: Exception

The ADB lock could not be acquired.

aio_androidtv.firetv module

Communicate with an Amazon Fire TV device via ADB over a network.

ADB Debugging must be enabled.

class aio_androidtv.firetv.**FireTV**(*host*, *port=5555*, *adbkey=''*, *state_detection_rules=None*)
Bases: *aio_androidtv.basetv.BaseTV*

Representation of an Amazon Fire TV device.

Parameters

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)
- **adbkey** (*str*) – The path to the adbkey file for ADB authentication
- **state_detection_rules** (*dict, None*) – A dictionary of rules for determining the state (see *BaseTV*)

DEVICE_CLASS = 'firetv'

async get_properties(*get_running_apps=True*, *lazy=False*)
Get the properties needed for Home Assistant updates.

This will send one of the following ADB commands:

- *aio_androidtv.constants.CMD_FIRETV_PROPERTIES_LAZY_RUNNING_APPS*
- *aio_androidtv.constants.CMD_FIRETV_PROPERTIES_LAZY_NO_RUNNING_APPS*
- *aio_androidtv.constants.CMD_FIRETV_PROPERTIES_NOT_LAZY_RUNNING_APPS*
- *aio_androidtv.constants.CMD_FIRETV_PROPERTIES_NOT_LAZY_NO_RUNNING_APPS*

Parameters

- **get_running_apps** (*bool*) – Whether or not to get the *running_apps()* property
- **lazy** (*bool*) – Whether or not to continue retrieving properties if the device is off or the screensaver is running

Returns

- **screen_on** (*bool, None*) – Whether or not the device is on, or `None` if it was not determined
- **awake** (*bool, None*) – Whether or not the device is awake (screensaver is not running), or `None` if it was not determined
- **wake_lock_size** (*int, None*) – The size of the current wake lock, or `None` if it was not determined
- **current_app** (*str, None*) – The current app property, or `None` if it was not determined
- **media_session_state** (*int, None*) – The state from the output of `dumpsys media_session`, or `None` if it was not determined
- **running_apps** (*list, None*) – A list of the running apps, or `None` if it was not determined

async get_properties_dict(*get_running_apps=True*, *lazy=True*)
Get the properties needed for Home Assistant updates and return them as a dictionary.

Parameters

- **get_running_apps** (*bool*) – Whether or not to get the *running_apps()* property
- **lazy** (*bool*) – Whether or not to continue retrieving properties if the device is off or the screensaver is running

Returns A dictionary with keys `'screen_on'`, `'awake'`, `'wake_lock_size'`, `'current_app'`, `'media_session_state'`, and `'running_apps'`

Return type dict

async running_apps()
Return a list of running user applications.

Returns A list of the running apps

Return type list

async turn_off()
Send `SLEEP` action if the device is not off.

async turn_on()
Send `POWER` and `HOME` actions if the device is off.

async update(*get_running_apps=True*)
Get the info needed for a Home Assistant update.

Parameters **get_running_apps** (*bool*) – Whether or not to get the *running_apps()* property

Returns

- **state** (*str*) – The state of the device
- **current_app** (*str*) – The current running app
- **running_apps** (*list*) – A list of the running apps if `get_running_apps` is True, otherwise the list `[current_app]`

1.1.2 Module contents

Connect to a device and determine whether it's an Android TV or an Amazon Fire TV.

ADB Debugging must be enabled.

aio_androidtv.**ha_state_detection_rules_validator**(*exc*)

Validate the rules (i.e., the `state_detection_rules` value) for a given app ID (i.e., a key in `state_detection_rules`).

See *BaseTV* for more info about the `state_detection_rules` parameter.

Parameters **exc** (*Exception*) – The exception that will be raised if a rule is invalid

Returns **wrapped_state_detection_rules_validator** – A function that is the same as *state_detection_rules_validator()*, but with the `exc` argument provided

Return type function

async aio_androidtv.**setup**(*host*, *port=5555*, *adbkey=''*, *state_detection_rules=None*, *device_class='auto'*, *auth_timeout_s=0.1*)

Connect to a device and determine whether it's an Android TV or an Amazon Fire TV.

Parameters

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)
- **adbkey** (*str*) – The path to the `adbkey` file for ADB authentication
- **adb_server_ip** (*str*) – The IP address of the ADB server
- **adb_server_port** (*int*) – The port for the ADB server
- **state_detection_rules** (*dict, None*) – A dictionary of rules for determining the state (see *BaseTV*)
- **device_class** (*str*) – The type of device: `'auto'` (detect whether it is an Android TV or Fire TV device), `'androidtv'`, or `'firetv'``
- **auth_timeout_s** (*float*) – Authentication timeout (in seconds)

Returns **aftv** – The representation of the device

Return type *AndroidTV*, *FireTV*

`aio-androidtv` is a Python 3.7+ package that provides state information and control of Android TV and Fire TV devices via ADB.

CHAPTER

TWO

INSTALLATION

Be sure you install into a Python 3.7+ environment.

```
pip install aio-androidtv
```

CHAPTER

THREE

ADB INTENTS AND COMMANDS

A collection of useful intents and commands can be found here (credit: mcfrojd).

CHAPTER

FOUR

ACKNOWLEDGMENTS

This is based on python-firetv by happyleavesaoc and the androidtv component for Home Assistant by a1ex4, and it depends on the Python package aio-adb-shell (which is based on python-adb).

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

P

R

S

T

U

V

W