

---

# **androidtv Documentation**

***Release 0.0.55***

**Jeff Irion**

**Nov 29, 2020**



# CONTENTS

<b>1</b>	<b>ADB Setup</b>	<b>1</b>
1.1	1. ADB Server . . . . .	1
1.2	2. Python ADB Implementation . . . . .	3
<b>2</b>	<b>androidtv</b>	<b>5</b>
2.1	androidtv package . . . . .	5
<b>3</b>	<b>Installation</b>	<b>49</b>
<b>4</b>	<b>ADB Intents and Commands</b>	<b>51</b>
<b>5</b>	<b>Acknowledgments</b>	<b>53</b>
<b>6</b>	<b>Indices and tables</b>	<b>55</b>
	<b>Python Module Index</b>	<b>57</b>
	<b>Index</b>	<b>59</b>



## ADB SETUP

This package works by sending ADB commands to your Android TV / Fire TV device. There are two ways to accomplish this.

### 1.1 1. ADB Server

`androidtv` can use a running ADB server to send ADB commands (credit: [pure-python-adb](#)). More info about ADB can be found here: [Android Debug Bridge \(adb\)](#). There are 3 main ways to setup an ADB server.

**Note:** The ADB server must be connected to your device(s) before starting Home Assistant. Otherwise, the components will not be setup.

#### 1.1.1 1a) Hass.io ADB Addon

For Hass.io users, this is the easiest option. Information about the addon can be found here: [Community Hass.io Add-ons: Android Debug Bridge](#). The configuration for the addon will look like:

```
{
  "log_level": "info",
  "devices": [
    "192.168.0.111",
    "192.168.0.222"
  ],
  "reconnect_timeout": 90,
  "keys_path": "/config/.android"
}
```

Your Home Assistant configuration will look like:

```
media_player:
- platform: androidtv
  name: Android TV 1
  host: 192.168.0.111
  adb_server_ip: 127.0.0.1

media_player:
- platform: androidtv
  name: Android TV 2
  host: 192.168.0.222
  adb_server_ip: 127.0.0.1
```

### 1.1.2 1b) Docker Container

Since Home Assistant isn't able to start the connection with the Android device directly, the ADB Server must do it instead. The ADB Server **must already be connected** to the Android device when Home Assistant attempts to access the ADB Server, or else Home Assistant will be unable to set up the Android device.

A modified script provided on the Home Assistant forums ([source](#)) demonstrates an example startup script for a Docker container that will automatically attempt, and continue to connect to a device when run:

Listing 1: `startup.sh`

```
#!/bin/sh

# for a single device, use: DEVICES=("192.168.0.111")
DEVICES=("192.168.0.111" "192.168.0.222")

echo "Starting up ADB..."

while true; do
  adb -a server nodaemon > /dev/null 2>&1
  sleep 10
done &

echo "Server started. Waiting for 30 seconds..."
sleep 30

echo "Connecting to devices."
for device in ${DEVICES[@]}; do
  adb connect $device
done
echo "Done."

while true; do
  for device in ${DEVICES[@]}; do
    adb connect $device > /dev/null 2>&1
  done
  sleep 60
done
```

Assuming the address of the ADB server is 192.168.0.101, your Home Assistant configuration will look like:

```
media_player:
- platform: androidtv
  name: Android TV 1
  host: 192.168.0.111
  adb_server_ip: 192.168.0.101

media_player:
- platform: androidtv
  name: Android TV 2
  host: 192.168.0.222
  adb_server_ip: 192.168.0.101
```

### 1.1.3 1c) Linux Service

TODO

Your Home Assistant configuration will look like:

```
media_player:
- platform: androidtv
  name: Android TV 1
  host: 192.168.0.111
  adb_server_ip: 127.0.0.1

media_player:
- platform: androidtv
  name: Android TV 2
  host: 192.168.0.222
  adb_server_ip: 127.0.0.1
```

## 1.2 2. Python ADB Implementation

The second way that `androidtv` can communicate with devices is using the Python ADB implementation (credit: `adb-shell`).

If your device requires ADB authentication, you will need to follow the instructions in the “ADB Authentication” section below. Once you have an authenticated key, this approach does not require any additional setup or addons. However, users with newer devices may find that the ADB connection is unstable. For a Fire TV device, you can try setting the `get_sources` configuration option to `false`. If the problem cannot be resolved, you should use the ADB server option.

Assuming you have 2 devices that require authentication, your configuration will look like this (update the `adbkey` path accordingly):

```
media_player:
- platform: androidtv
  name: Android TV 1
  host: 192.168.0.111
  adbkey: "/config/.android/adbkey"

media_player:
- platform: androidtv
  name: Android TV 2
  host: 192.168.0.222
  adbkey: "/config/.android/adbkey"
```

### 1.2.1 ADB Authentication

If you get a “Device authentication required, no keys available” error when trying to set up your Android TV or Fire TV, then you’ll need to create an `adbkey` and add its path to your configuration. Follow the instructions on this page to connect to your device from your computer: [Connecting to Fire TV Through adb](#).

---

**Note:** In the dialog appearing on your Android TV / Fire TV, you must check the box that says “always allow connections from this device.” ADB authentication in Home Assistant will only work using a trusted key.

---

Once you’ve successfully connected to your Android TV / Fire TV via the command `adb connect <ipaddress>`, the file `adbkey` will be created on your computer. The default location for this file is (from <https://developer.android.com/studio/command-line/adb>):

- Linux and Mac: `$HOME/.android`
- Windows: `%userprofile%\.android`

Copy the `adbkey` file to your Home Assistant folder and add the path to your configuration.



## 2.1 androidtv package

### 2.1.1 Subpackages

**androidtv.adb\_manager package**

**Submodules**

**androidtv.adb\_manager.adb\_manager\_async module**

Classes to manage ADB connections.

- *ADBPYthonAsync* utilizes a Python implementation of the ADB protocol.
- *ADBServerAsync* utilizes an ADB server to communicate with the device.

```
class androidtv.adb_manager.adb_manager_async.ADBPYthonAsync (host, port,  
                                                             adbkey=",  
                                                             signer=None)
```

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
- **signer** (*PythonRSASigner*, *None*) – The signer for the ADB keys, as loaded by *ADBPYthonAsync.load\_adbkey()*

#### 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=10.0*)

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

**static load\_adbkey** (*adbkey*)

Load the ADB keys.

**Parameters** **adbkey** (*str*) – The path to the `adbkey` file for ADB authentication

**Returns** The `PythonRSASigner` with the key files loaded

**Return type** `PythonRSASigner`

**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

```
class androidtv.adb_manager.adb_manager_async.ADBServerAsync (host, port=5555,
                                                             adb_server_ip="",
                                                             adb_server_port=5037)
```

Bases: object

A manager for ADB connections that uses an ADB server.

#### 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)
- **adb\_server\_ip** (*str*) – The IP address of the ADB server

- **adb\_server\_port** (*int*) – The port for the ADB server

**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 server socket connection.

Currently, this doesn't do anything except set `self._available = False`.

**async connect** (*always\_log\_errors=True*)

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

**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 an ADB server.

**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 an ADB server.

**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 screenshot** ()

Take a screenshot using an ADB server.

**Returns** The screenshot as a binary .png image, or None if there was an `IndexError` exception

**Return type** bytes, None

**async shell** (*cmd*)

Send an ADB command using an ADB server.

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

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

**Return type** str, None

**class** androidtv.adb\_manager.adb\_manager\_async.**ClientAsync** (*host, port*)

Bases: object

An async wrapper for the pure-python-adb `Client` class.

**async device** (*serial*)

Get a `DeviceAsync` instance.

**class** androidtv.adb\_manager.adb\_manager\_async.**DeviceAsync** (*device*)

Bases: object

An async wrapper for the pure-python-adb Device class.

**async pull** (*device\_path*, *local\_path*)

Download a file.

**async push** (*local\_path*, *device\_path*)

Upload a file.

**async screencap** ()

Take a screencap.

**async shell** (*cmd*)

Send a shell command.

androidtv.adb\_manager.adb\_manager\_async.**\_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

## androidtv.adb\_manager.adb\_manager\_sync module

Classes to manage ADB connections.

- **ADBPYthonSync** utilizes a Python implementation of the ADB protocol.
- **ADBServerSync** utilizes an ADB server to communicate with the device.

**class** androidtv.adb\_manager.adb\_manager\_sync.**ADBPYthonSync** (*host*, *port*, *adbkey=""*,  
*signer=None*)

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
- **signer** (*PythonRSASigner*, *None*) – The signer for the ADB keys, as loaded by `ADBPYthonSync.load_adbkey()`

#### property available

Check whether the ADB connection is intact.

**Returns** Whether or not the ADB connection is intact

**Return type** bool

**close** ()

Close the ADB socket connection.

**connect** (*always\_log\_errors=True, auth\_timeout\_s=10.0*)

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

**static load\_adbkey** (*adbkey*)

Load the ADB keys.

**Parameters** **adbkey** (*str*) – The path to the adbkey file for ADB authentication

**Returns** The PythonRSASigner with the key files loaded

**Return type** PythonRSASigner

**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

**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

**screencap** ()

Take a screenshot using the Python ADB implementation.

**Returns** The screencap as a binary .png image

**Return type** bytes

**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

```
class androidtv.adb_manager.adb_manager_sync.ADBServerSync (host, port=5555,  
                                                         adb_server_ip=  
                                                         adb_server_port=5037)
```

Bases: object

A manager for ADB connections that uses an ADB server.

**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)

- **adb\_server\_ip** (*str*) – The IP address of the ADB server
- **adb\_server\_port** (*int*) – The port for the ADB server

**property available**

Check whether the ADB connection is intact.

**Returns** Whether or not the ADB connection is intact

**Return type** bool

**close()**

Close the ADB server socket connection.

Currently, this doesn't do anything except set `self._available = False`.

**connect** (*always\_log\_errors=True*)

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

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

**Return type** bool

**pull** (*local\_path, device\_path*)

Pull a file from the device using an ADB server.

**Parameters**

- **local\_path** (*str*) – The path where the file will be saved
- **device\_path** (*str*) – The file on the device that will be pulled

**push** (*local\_path, device\_path*)

Push a file to the device using an ADB server.

**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

**screenshot()**

Take a screenshot using an ADB server.

**Returns** The screenshot as a binary .png image, or None if there was an `IndexError` exception

**Return type** bytes, None

**shell** (*cmd*)

Send an ADB command using an ADB server.

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

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

**Return type** str, None

`androidtv.adb_manager.adb_manager_sync.LOCK_KWARGS = {'timeout': 3.0}`

Use a timeout for the ADB threading lock if it is supported

`androidtv.adb_manager.adb_manager_sync._acquire(lock)`

Handle acquisition and release of a `threading.Lock` object with `LOCK_KWARGS` keyword arguments.

**Parameters** **lock** (*threading.Lock*) – The lock that we will try to acquire

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

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

## Module contents

### androidtv.androidtv package

#### Submodules

#### androidtv.androidtv.androidtv\_async module

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

ADB Debugging must be enabled.

```
class androidtv.androidtv.androidtv_async.AndroidTVAsync (host, port=5555,
                                                         adbkey="",
                                                         adb_server_ip="",
                                                         adb_server_port=5037,
                                                         state_detection_rules=None,
                                                         signer=None)
Bases:      androidtv.basetv.basetv_async.BaseTVAsync,      androidtv.androidtv.
            base_androidtv.BaseAndroidTV
```

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
- **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*)
- **signer** (*PythonRSASigner*, *None*) – The signer for the ADB keys, as loaded by `androidtv.adb_manager.adb_manager_async.ADBPythonAsync.load_adbkey()`

**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:

- `androidtv.constants.CMD_ANDROIDTV_PROPERTIES_LAZY_RUNNING_APPS`
- `androidtv.constants.CMD_ANDROIDTV_PROPERTIES_LAZY_NO_RUNNING_APPS`
- `androidtv.constants.CMD_ANDROIDTV_PROPERTIES_NOT_LAZY_RUNNING_APPS`
- `androidtv.constants.CMD_ANDROIDTV_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
- **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
- **hdmi\_input** (*str, None*) – The HDMI input, or *None* if it could not be 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', 'running\_apps', and 'hdmi\_input'

**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, lazy=True*)

Get the info needed for a Home Assistant update.



**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**

- **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)

**androidtv.androidtv.androidtv\_sync module**

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

ADB Debugging must be enabled.

```
class androidtv.androidtv.androidtv_sync.AndroidTVSync (host, port=5555, adbkey=", adb_server_ip",  
                                                    adb_server_port=5037,  
                                                    state_detection_rules=None,  
                                                    signer=None)
```

Bases: `androidtv.basetv.basetv_sync.BaseTVSync`, `androidtv.androidtv.base_androidtv.BaseAndroidTV`

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
- **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`)
- **signer** (*PythonRSASigner*, *None*) – The signer for the ADB keys, as loaded by `androidtv.adb_manager.adb_manager_sync.ADBPythonSync.load_adbkey()`

**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:

- `androidtv.constants.CMD_ANDROIDTV_PROPERTIES_LAZY_RUNNING_APPS`
- `androidtv.constants.CMD_ANDROIDTV_PROPERTIES_LAZY_NO_RUNNING_APPS`

- `androidtv.constants.CMD_ANDROIDTV_PROPERTIES_NOT_LAZY_RUNNING_APPS`
- `androidtv.constants.CMD_ANDROIDTV_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
- **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
- **hdmi\_input** (*str, None*) – The HDMI input, or *None* if it could not be determined

**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', 'running\_apps', and 'hdmi\_input'

**Return type** dict

**running\_apps** ()

Return a list of running user applications.

**Returns** A list of the running apps

**Return type** list

**turn\_off()**

Send POWER action if the device is not off.

**turn\_on()**

Send POWER action if the device is off.

**update** (*get\_running\_apps=True, lazy=True*)

Get the info needed for a Home Assistant update.

#### 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

- **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)
- **hdmi\_input** (*str, None*) – The HDMI input, or None if it could not be determined

## androidtv.androidtv.base\_androidtv module

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

ADB Debugging must be enabled.

```
class androidtv.androidtv.base_androidtv.BaseAndroidTV(host, port=5555, adbkey=", adb_server_ip=", adb_server_port=5037, state_detection_rules=None)
```

Bases: `androidtv.basetv.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
- **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 = 'androidtv'
```

**\_get\_properties** (*output, get\_running\_apps*)

Get the properties needed for Home Assistant updates.

**Parameters**

- **output** (*str*, *None*) – The output of the ADB command used to retrieve the properties
- **get\_running\_apps** (*bool*) – Whether or not to get the `running_apps` property

**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
- **hdmi\_input** (*str*, *None*) – The HDMI input, or *None* if it could not be determined

**\_update** (*screen\_on*, *awake*, *audio\_state*, *wake\_lock\_size*, *current\_app*, *media\_session\_state*, *audio\_output\_device*, *is\_volume\_muted*, *volume*, *running\_apps*, *hdmi\_input*)  
Get the info needed for a Home Assistant update.

**Parameters**

- **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
- **hdmi\_input** (*str*, *None*) – The HDMI input, or *None* if it could not be determined

## 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)
- **hdmi\_input** (*str, None*) – The HDMI input, or `None` if it could not be determined

## Module contents

## androidtv.basetv package

## Submodules

## androidtv.basetv.basetv module

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

ADB Debugging must be enabled.

```
class androidtv.basetv.basetv.BaseTV(adb, host, port=5555, adbkey="", adb_server_ip="",
                                       adb_server_port=5037, 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

- **adb** (`ADBPythonSync`, `ADBServerSync`, `ADBPythonAsync`, `ADBServerAsync`) – The handler for ADB commands
- **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 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 `androidtv.constants.CMD_AUDIO_STATE`.

**Parameters** **audio\_state\_response** (`str`, `None`) – The output of the command `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 `androidtv.constants.CMD_CURRENT_APP`.

**Parameters** **current\_app\_response** (*str*, *None*) – The output from the ADB command `androidtv.constants.CMD_CURRENT_APP`

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

**Return type** `str`, `None`

**\_current\_app\_media\_session\_state**(*media\_session\_state\_response*)

Get the current app and the media session state properties from the output of `androidtv.constants.CMD_MEDIA_SESSION_STATE_FULL`.

**Parameters** **media\_session\_state\_response** (*str*, *None*) – The output of `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`

**static** **\_get\_hdmi\_input**(*hdmi\_response*)

Get the HDMI input from the output of `androidtv.constants.CMD_HDMI_INPUT`.

**Parameters** **hdmi\_response** (*str*, *None*) – The output of `androidtv.constants.CMD_HDMI_INPUT`

**Returns** The HDMI input, or `None` if it could not be determined

**Return type** `str`, `None`

**static** **\_installed\_apps**(*installed\_apps\_response*)

Get the installed apps from the output of `androidtv.constants.CMD_INSTALLED_APPS`.

**Parameters** **installed\_apps\_response** (*str*, *None*) – The output of `androidtv.constants.CMD_INSTALLED_APPS`

**Returns** A list of the installed apps, or None if it could not be determined

**Return type** list, 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

**static** `_media_session_state(media_session_state_response, current_app)`

Get the state from the output of `androidtv.constants.CMD_MEDIA_SESSION_STATE`.

**Parameters**

- `media_session_state_response` (*str*, None) – The output of `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_device_properties(properties)`

Return a dictionary of device properties.

**Parameters** `properties` (*str*, None) – The output of the ADB command that retrieves the device properties

**Returns** A dictionary with keys 'wifimac', 'ethmac', 'serialno', 'manufacturer', 'model', and 'sw\_version'

**Return type** dict

**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** `_parse_stream_music(stream_music_raw)`

Parse the output of the command `androidtv.constants.CMD_STREAM_MUSIC`.

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

**Returns** The STREAM\_MUSIC block from the output of `androidtv.constants.CMD_STREAM_MUSIC`, or None if it could not be determined

**Return type** str, None

**static** `_running_apps(running_apps_response)`

Get the running apps from the output of `androidtv.constants.CMD_RUNNING_APPS`.

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



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

**Return type** list, None

**\_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 `androidtv.constants.CMD_WAKE_LOCK_SIZE`.

**Parameters** **wake\_lock\_size\_response** (*str*, None) – The output of `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

**property available**

Whether the ADB connection is intact.

**Returns** Whether or not the ADB connection is intact

**Return type** bool

`androidtv.basetv.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*

## androidtv.basetv.basetv\_async module

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

ADB Debugging must be enabled.

```
class androidtv.basetv.basetv_async.BaseTVAsync (host, port=5555, adb_server_ip=",  
                                              adb_server_port=5037,  
                                              state_detection_rules=None,  
                                              signer=None)
```

Bases: *androidtv.basetv.basetv.BaseTV*

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
- **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 above)
- **signer** (*PythonRSASigner*, *None*) – The signer for the ADB keys, as loaded by `androidtv.adb_manager.adb_manager_async.ADBPythonAsync.load_adbkey()`

**async \_get\_stream\_music** (*stream\_music\_raw=None*)

Get the STREAM\_MUSIC block from the output of the command `androidtv.constants.CMD_STREAM_MUSIC`.

**Parameters** **stream\_music\_raw** (*str*, *None*) – The output of the command `androidtv.constants.CMD_STREAM_MUSIC`

**Returns** The STREAM\_MUSIC block from the output of `androidtv.constants.CMD_STREAM_MUSIC`, or *None* if it could not be determined

**Return type** *str*, *None*

**async \_key** (*key*)

Send a key event to device.

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

**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*

**async adb\_close** ()

Close the ADB connection.

This only works for the Python ADB implementation (see `androidtv.adb_manager.adb_manager_async.ADBPython.close()`). For the ADB server approach, this doesn't do anything (see `androidtv.adb_manager.adb_manager_async.ADBServer.close()`).

**async adb\_connect** (*always\_log\_errors=True*, *auth\_timeout\_s=10.0*)

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 `androidtv.adb_manager.adb_manager_async.ADBPythonAsync.pull()` or `androidtv.adb_manager.adb_manager_async.ADBServerAsync.pull()`, depending on whether the Python ADB implementation or an ADB server is used for communicating with the device.

**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 `androidtv.adb_manager.adb_manager_async.ADBPythonAsync.push()` or `androidtv.adb_manager.adb_manager_async.ADBServerAsync.push()`, depending on whether the Python ADB implementation or an ADB server is used for communicating with the device.

**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\_screenshot** ()

Take a screenshot.

This calls `androidtv.adb_manager.adb_manager_async.ADBPythonAsync.screenshot()` or `androidtv.adb_manager.adb_manager_async.ADBServerAsync.screenshot()`, depending on whether the Python ADB implementation or an ADB server is used for communicating with the device.

**Returns** The screenshot as a binary .png image

**Return type** bytes

**async adb\_shell** (*cmd*)

Send an ADB command.

This calls `androidtv.adb_manager.adb_manager_async.ADBPythonAsync.shell()` or `androidtv.adb_manager.adb_manager_async.ADBServerAsync.shell()`, depending on whether the Python ADB implementation or an ADB server is used for communicating with the device.

**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 `androidtv.constants.CMD_AUDIO_STATE`, or `None` if it could not be determined

**Return type** str, None

**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 get\_hdmi\_input()**

Get the HDMI input from the output of `androidtv.constants.CMD_HDMI_INPUT`.

**Returns** The HDMI input, or `None` if it could not be determined

**Return type** str, None

**async get\_installed\_apps()**

Return a list of installed applications.

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

**Return type** list, None

**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`

## androidtv.basetv.basetv\_sync module

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

ADB Debugging must be enabled.

```
class androidtv.basetv.basetv_sync.BaseTVSync (host, port=5555, adbkey=adb_server_ip=adb_server_port=5037,
                                             state_detection_rules=None,
                                             signer=None)
```

Bases: *androidtv.basetv.basetv.BaseTV*

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
- **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 above)
- **signer** (*PythonRSASigner*, *None*) – The signer for the ADB keys, as loaded by *androidtv.adb\_manager.adb\_manager\_sync.ADBPythonSync.load\_adbkey()*

**`_get_stream_music`** (*stream\_music\_raw=None*)

Get the STREAM\_MUSIC block from the output of the command `androidtv.constants.CMD_STREAM_MUSIC`.

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

**Returns** The STREAM\_MUSIC block from the output of `androidtv.constants.CMD_STREAM_MUSIC`, or None if it could not be determined

**Return type** `str, None`

**`_key`** (*key*)

Send a key event to device.

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

**`_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`

**`adb_close`** ()

Close the ADB connection.

This only works for the Python ADB implementation (see `androidtv.adb_manager.adb_manager_sync.ADBPythonSync.close()`). For the ADB server approach, this doesn't do anything (see `androidtv.adb_manager.adb_manager_sync.ADBServerSync.close()`).

**`adb_connect`** (*always\_log\_errors=True, auth\_timeout\_s=10.0*)

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`

**`adb_pull`** (*local\_path, device\_path*)

Pull a file from the device.

This calls `androidtv.adb_manager.adb_manager_sync.ADBPythonSync.pull()` or `androidtv.adb_manager.adb_manager_sync.ADBServerSync.pull()`, depending on whether the Python ADB implementation or an ADB server is used for communicating with the device.

**Parameters**

- **local\_path** (*str*) – The path where the file will be saved
- **device\_path** (*str*) – The file on the device that will be pulled

**adb\_push** (*local\_path*, *device\_path*)

Push a file to the device.

This calls `androidtv.adb_manager.adb_manager_sync.ADBPythonSync.push()` or `androidtv.adb_manager.adb_manager_sync.ADBServerSync.push()`, depending on whether the Python ADB implementation or an ADB server is used for communicating with the device.

**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

**adb\_screenshot** ()

Take a screenshot.

This calls `androidtv.adb_manager.adb_manager_sync.ADBPythonSync.screenshot()` or `androidtv.adb_manager.adb_manager_sync.ADBServerSync.screenshot()`, depending on whether the Python ADB implementation or an ADB server is used for communicating with the device.

**Returns** The screenshot as a binary .png image

**Return type** bytes

**adb\_shell** (*cmd*)

Send an ADB command.

This calls `androidtv.adb_manager.adb_manager_sync.ADBPythonSync.shell()` or `androidtv.adb_manager.adb_manager_sync.ADBServerSync.shell()`, depending on whether the Python ADB implementation or an ADB server is used for communicating with the device.

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

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

**Return type** str, None

**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

**audio\_state** ()

Check if audio is playing, paused, or idle.

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

**Return type** str, None

**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

**back** ()

Send back action.

**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`

**down()**

Send down action.

**enter()**

Send enter action.

**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

**get\_hdmi\_input()**

Get the HDMI input from the output of `androidtv.constants.CMD_HDMI_INPUT`.

**Returns** The HDMI input, or `None` if it could not be determined

**Return type** str, `None`

**get\_installed\_apps()**

Return a list of installed applications.

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

**Return type** list, `None`

**home()**

Send home action.

**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`

**key\_0()**

Send 0 keypress.

**key\_1()**

Send 1 keypress.

**key\_2()**

Send 2 keypress.

**key\_3()**

Send 3 keypress.

**key\_4()**

Send 4 keypress.

**key\_5()**

Send 5 keypress.

**key\_6()**

Send 6 keypress.

**key\_7()**  
Send 7 keypress.

**key\_8()**  
Send 8 keypress.

**key\_9()**  
Send 9 keypress.

**key\_a()**  
Send a keypress.

**key\_b()**  
Send b keypress.

**key\_c()**  
Send c keypress.

**key\_d()**  
Send d keypress.

**key\_e()**  
Send e keypress.

**key\_f()**  
Send f keypress.

**key\_g()**  
Send g keypress.

**key\_h()**  
Send h keypress.

**key\_i()**  
Send i keypress.

**key\_j()**  
Send j keypress.

**key\_k()**  
Send k keypress.

**key\_l()**  
Send l keypress.

**key\_m()**  
Send m keypress.

**key\_n()**  
Send n keypress.

**key\_o()**  
Send o keypress.

**key\_p()**  
Send p keypress.

**key\_q()**  
Send q keypress.

**key\_r()**  
Send r keypress.

**key\_s()**  
Send s keypress.

**key\_t()**  
Send t keypress.

**key\_u()**  
Send u keypress.

**key\_v()**  
Send v keypress.

**key\_w()**  
Send w keypress.

**key\_x()**  
Send x keypress.

**key\_y()**  
Send y keypress.

**key\_z()**  
Send z keypress.

**launch\_app(app)**  
Launch an app.

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

**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`

**left()**  
Send left action.

**media\_next\_track()**  
Send media next action (results in fast-forward).

**media\_pause()**  
Send media pause action.

**media\_play()**  
Send media play action.

**media\_play\_pause()**  
Send media play/pause action.

**media\_previous\_track()**  
Send media previous action (results in rewind).

**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`

**media\_stop()**  
Send media stop action.

**menu()**  
Send menu action.

**mute\_volume()**  
Mute the volume.

**power()**  
Send power action.

**right()**  
Send right action.

**screen\_on()**  
Check if the screen is on.

**Returns** Whether or not the device is on

**Return type** `bool`

**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`

**sleep()**  
Send sleep action.

**space()**  
Send space keypress.

**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>`

**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`

**up()**  
Send up action.

**volume()**  
Get the absolute volume level.

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



**Return type** int, None

**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

**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

**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

**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

## Module contents

### androidtv.firetv package

#### Submodules

#### androidtv.firetv.base\_firetv module

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

ADB Debugging must be enabled.

```
class androidtv.firetv.base_firetv.BaseFireTV(host,          port=5555,          adb-
                                             bkey="",          adb_server_ip="",
                                             adb_server_port=5037,
                                             state_detection_rules=None)
```

Bases: [androidtv.basetv.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
- **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** = 'firetv'

**\_get\_properties** (*output*, *get\_running\_apps=True*)

Get the properties needed for Home Assistant updates.

This will send one of the following ADB commands:

- *androidtv.constants.CMD\_FIRETV\_PROPERTIES\_LAZY\_RUNNING\_APPS*
- *androidtv.constants.CMD\_FIRETV\_PROPERTIES\_LAZY\_NO\_RUNNING\_APPS*
- *androidtv.constants.CMD\_FIRETV\_PROPERTIES\_NOT\_LAZY\_RUNNING\_APPS*
- *androidtv.constants.CMD\_FIRETV\_PROPERTIES\_NOT\_LAZY\_NO\_RUNNING\_APPS*

#### Parameters

- **output** (*str*, *None*) – The output of the ADB command used to retrieve the properties
- **get\_running\_apps** (*bool*) – Whether or not to get the *running\_apps* property

#### 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
- **hdmi\_input** (*str*, *None*) – The HDMI input, or *None* if it could not be determined

**\_update** (*screen\_on*, *awake*, *wake\_lock\_size*, *current\_app*, *media\_session\_state*, *running\_apps*, *hdmi\_input*)

Get the info needed for a Home Assistant update.

#### Parameters

- **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
- **hdmi\_input** (*str*, *None*) – The HDMI input, or *None* if it could not be determined

#### 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`]
- **hdmi\_input** (*str*, *None*) – The HDMI input, or *None* if it could not be determined

### androidtv.firetv.firetv\_async module

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

ADB Debugging must be enabled.

```
class androidtv.firetv.firetv_async.FireTVAsync (host, port=5555, adbkey=",", adb_server_ip=",",
                                              adb_server_port=5037,
                                              state_detection_rules=None,
                                              signer=None)
```

Bases: `androidtv.basetv.basetv_async.BaseTVAsync`, `androidtv.firetv.base_firetv.BaseFireTV`

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
- **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`)
- **signer** (`PythonRSASigner`, *None*) – The signer for the ADB keys, as loaded by `androidtv.adb_manager.adb_manager_async.ADBPythonAsync.load_adbkey()`

**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:

- `androidtv.constants.CMD_FIRETV_PROPERTIES_LAZY_RUNNING_APPS`
- `androidtv.constants.CMD_FIRETV_PROPERTIES_LAZY_NO_RUNNING_APPS`

- `androidtv.constants.CMD_FIRETV_PROPERTIES_NOT_LAZY_RUNNING_APPS`
- `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
- **hdmi\_input** (*str, None*) – The HDMI input, or *None* if it could not be 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', 'running\_apps', and 'hdmi\_input'

**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, lazy=True*)

Get the info needed for a Home Assistant update.

#### 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

- **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]`
- **hdmi\_input** (*str, None*) – The HDMI input, or None if it could not be determined

## androidtv.firetv.firetv\_sync module

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

ADB Debugging must be enabled.

```
class androidtv.firetv.firetv_sync.FireTVSync(host, port=5555, adb=ad-
                                             bkey=", adb_server_ip=",
                                             adb_server_port=5037,
                                             state_detection_rules=None,
                                             signer=None)
```

Bases: `androidtv.basetv.basetv_sync.BaseTVSync`, `androidtv.firetv.base_firetv.BaseFireTV`

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
- **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`)
- **signer** (`PythonRSASigner, None`) – The signer for the ADB keys, as loaded by `androidtv.adb_manager.adb_manager_sync.ADBPythonSync.load_adbkey()`

**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:

- `androidtv.constants.CMD_FIRETV_PROPERTIES_LAZY_RUNNING_APPS`
- `androidtv.constants.CMD_FIRETV_PROPERTIES_LAZY_NO_RUNNING_APPS`
- `androidtv.constants.CMD_FIRETV_PROPERTIES_NOT_LAZY_RUNNING_APPS`
- `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
- **hdmi\_input** (*str, None*) – The HDMI input, or *None* if it could not be determined

**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', 'running\_apps', and 'hdmi\_input'

**Return type** dict

**running\_apps** ()

Return a list of running user applications.

**Returns** A list of the running apps

**Return type** list

**turn\_off** ()

Send SLEEP action if the device is not off.

**turn\_on** ()

Send POWER and HOME actions if the device is off.

**update** (*get\_running\_apps=True, lazy=True*)

Get the info needed for a Home Assistant update.

**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**

- **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]`
- **hdmi\_input** (*str, None*) – The HDMI input, or `None` if it could not be determined

## Module contents

### 2.1.2 Submodules

#### androidtv.constants module

Constants used throughout the code.

#### Links

- [ADB key event codes](#)
- [MediaSession PlaybackState property](#)

```
androidtv.constants.CMD_ANDROIDTV_PROPERTIES_LAZY_NO_RUNNING_APPS = "(dumpsys power | grep
    Get the properties for an Android TV device (lazy=True, get_running_apps=False); see
    androidtv.androidtv.androidtv_sync.AndroidTVSync.get_properties() and
    androidtv.androidtv.androidtv_async.AndroidTVAsync.get_properties()
```

```
androidtv.constants.CMD_ANDROIDTV_PROPERTIES_LAZY_RUNNING_APPS = "(dumpsys power | grep 'D
    Get the properties for an Android TV device (lazy=True, get_running_apps=True); see
    androidtv.androidtv.androidtv_sync.AndroidTVSync.get_properties() and
    androidtv.androidtv.androidtv_async.AndroidTVAsync.get_properties()
```

```
androidtv.constants.CMD_ANDROIDTV_PROPERTIES_NOT_LAZY_NO_RUNNING_APPS = "(dumpsys power |
    Get the properties for an Android TV device (lazy=False, get_running_apps=False); see
    androidtv.androidtv.androidtv_sync.AndroidTVSync.get_properties() and
    androidtv.androidtv.androidtv_async.AndroidTVAsync.get_properties()
```

```
androidtv.constants.CMD_ANDROIDTV_PROPERTIES_NOT_LAZY_RUNNING_APPS = "(dumpsys power | grep
    Get the properties for an Android TV device (lazy=False, get_running_apps=True); see
    androidtv.androidtv.androidtv_sync.AndroidTVSync.get_properties() and
    androidtv.androidtv.androidtv_async.AndroidTVAsync.get_properties()
```

```
androidtv.constants.CMD_AUDIO_STATE = "dumpsys audio | grep paused | grep -qv 'Buffer Queue
    Get the audio state
```

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

```
androidtv.constants.CMD_CURRENT_APP = 'CURRENT_APP=$(dumpsys window windows | grep mCurrent
    Get the current app
```

```
androidtv.constants.CMD_CURRENT_APP_GOOGLE_TV = "CURRENT_APP=$(dumpsys activity a . | grep
    Get the current app for a Google TV device
```

```
androidtv.constants.CMD_FIRETV_PROPERTIES_LAZY_NO_RUNNING_APPS = "(dumpsys power | grep 'D
    Get the properties for a Fire TV device (lazy=True, get_running_apps=False); see
    androidtv.firetv.firetv_sync.FireTVSync.get_properties() and androidtv.
    firetv.firetv_async.FireTVAsync.get_properties()
```

```
androidtv.constants.CMD_FIRETV_PROPERTIES_LAZY_RUNNING_APPS = "(dumpsys power | grep 'Disp
    Get the properties for a Fire TV device (lazy=True, get_running_apps=True); see androidtv.
    firetv.firetv_sync.FireTVSync.get_properties() and androidtv.firetv.
    firetv_async.FireTVAsync.get_properties()
```

```

androidtv.constants.CMD_FIRETV_PROPERTIES_NOT_LAZY_NO_RUNNING_APPS = "(dumpsys power | grep 'D
    Get the properties for a Fire TV device (lazy=False, get_running_apps=False); see
    androidtv.firetv.firetv_sync.FireTVSync.get_properties() and androidtv.
    firetv.firetv_async.FireTVAsync.get_properties()

androidtv.constants.CMD_FIRETV_PROPERTIES_NOT_LAZY_RUNNING_APPS = "(dumpsys power | grep 'D
    Get the properties for a Fire TV device (lazy=False, get_running_apps=True); see
    androidtv.firetv.firetv_sync.FireTVSync.get_properties() and androidtv.
    firetv.firetv_async.FireTVAsync.get_properties()

androidtv.constants.CMD_GOOGLE_TV_PROPERTIES_LAZY_NO_RUNNING_APPS = "(dumpsys power | grep 'D
    Get the properties for a Google TV device (lazy=True, get_running_apps=False); see
    androidtv.androidtv.androidtv_sync.AndroidTVSync.get_properties() and
    androidtv.androidtv.androidtv_async.AndroidTVAsync.get_properties()

androidtv.constants.CMD_GOOGLE_TV_PROPERTIES_LAZY_RUNNING_APPS = "(dumpsys power | grep 'D
    Get the properties for a Google TV device (lazy=True, get_running_apps=True); see
    androidtv.androidtv.androidtv_sync.AndroidTVSync.get_properties() and
    androidtv.androidtv.androidtv_async.AndroidTVAsync.get_properties()

androidtv.constants.CMD_GOOGLE_TV_PROPERTIES_NOT_LAZY_NO_RUNNING_APPS = "(dumpsys power | g
    Get the properties for a Google TV device (lazy=False, get_running_apps=False); see
    androidtv.androidtv.androidtv_sync.AndroidTVSync.get_properties() and
    androidtv.androidtv.androidtv_async.AndroidTVAsync.get_properties()

androidtv.constants.CMD_GOOGLE_TV_PROPERTIES_NOT_LAZY_RUNNING_APPS = "(dumpsys power | grep
    Get the properties for a Google TV device (lazy=False, get_running_apps=True); see
    androidtv.androidtv.androidtv_sync.AndroidTVSync.get_properties() and
    androidtv.androidtv.androidtv_async.AndroidTVAsync.get_properties()

androidtv.constants.CMD_HDMI_INPUT = "dumpsys activity starter | grep -o 'HDMIInputService'
    Get the HDMI input

androidtv.constants.CMD_INSTALLED_APPS = 'pm list packages'
    Get installed apps

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

androidtv.constants.CMD_LAUNCH_APP_GOOGLE_TV = "CURRENT_APP=$(dumpsys activity a . | grep -
    Launch an app if it is not already the current app (for Google TV devices)

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

androidtv.constants.CMD_MEDIA_SESSION_STATE_FULL = "CURRENT_APP=$(dumpsys window windows |
    Determine the current app and get the state from dumpsys media_session

androidtv.constants.CMD_RUNNING_APPS_ANDROIDTV = 'ps -A | grep u0_a'
    Get the running apps for an Android TV device

androidtv.constants.CMD_RUNNING_APPS_FIRETV = 'ps | grep u0_a'
    Get the running apps for a Fire TV device

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

androidtv.constants.CMD_STREAM_MUSIC = "dumpsys audio | grep '\\- STREAM_MUSIC:' -A 11"
    Get the "STREAM_MUSIC" block from dumpsys audio

```



```

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

androidtv.constants.DEFAULT_ADB_TIMEOUT_S = 9.0
    Default timeout (in s) for adb_shell.handle.tcp_handle.TcpHandle and adb_shell.handle.
    tcp_handle_async.TcpHandleAsync

androidtv.constants.DEFAULT_AUTH_TIMEOUT_S = 10.0
    Default authentication timeout (in s) for adb_shell.handle.tcp_handle.TcpHandle.connect()
    and adb_shell.handle.tcp_handle_async.TcpHandleAsync.connect()

androidtv.constants.DEFAULT_LOCK_TIMEOUT_S = 3.0
    Default timeout for acquiring the lock that protects ADB commands

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

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

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

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

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())

```

## androidtv.exceptions module

Exceptions for use throughout the code.

**exception** androidtv.exceptions.LockNotAcquiredException

Bases: Exception

The ADB lock could not be acquired.

## androidtv.setup\_async module

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

ADB Debugging must be enabled.

```

async androidtv.setup_async.setup(host, port=5555, adbkey="", adb_server_ip="",
                                   adb_server_port=5037, state_detection_rules=None,
                                   device_class='auto', auth_timeout_s=10.0, signer=None)

```

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)
- **signer** (*PythonRSASigner*, *None*) – The signer for the ADB keys, as loaded by `androidtv.adb_manager.adb_manager_async.ADBPythonAsync.load_adbkey()`

**Returns** The representation of the device

**Return type** [AndroidTVAsync](#), [FireTVAsync](#)

## 2.1.3 Module contents

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

ADB Debugging must be enabled.

`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

`androidtv.setup` (*host*, *port=5555*, *adbkey=""*, *adb\_server\_ip=""*, *adb\_server\_port=5037*, *state\_detection\_rules=None*, *device\_class='auto'*, *auth\_timeout\_s=10.0*, *signer=None*)

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)
- **signer** (*PythonRSASigner*, *None*) – The signer for the ADB keys, as loaded by `androidtv.adb_manager.adb_manager_sync.ADBPythonSync.load_adbkey()`

**Returns** The representation of the device

**Return type** *AndroidTVSync, FireTVSync*

`androidtv` is a Python package that provides state information and control of Android TV and Fire TV devices via ADB. This package is used by the [Android TV](#) integration in Home Assistant.



## INSTALLATION

```
pip install androidtv
```

To utilize the async version of this code, you must install into a Python 3.7+ environment via:

```
pip install androidtv[async]
```



## ADB INTENTS AND COMMANDS

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





## **ACKNOWLEDGMENTS**

This is based on [python-firetv](#) by happyleavesaoc and the [androidtv](#) component for [Home Assistant](#) by alex4, and it depends on the Python packages [adb-shell](#) (which is based on [python-adb](#)) and [pure-python-adb](#).



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### a

- `androidtv`, [46](#)
- `androidtv.adb_manager`, [11](#)
- `androidtv.adb_manager.adb_manager_async`,  
[5](#)
- `androidtv.adb_manager.adb_manager_sync`,  
[8](#)
- `androidtv.androidtv`, [17](#)
- `androidtv.androidtv.androidtv_async`, [11](#)
- `androidtv.androidtv.androidtv_sync`, [13](#)
- `androidtv.androidtv.base_androidtv`, [15](#)
- `androidtv.basetv`, [37](#)
- `androidtv.basetv.basetv`, [17](#)
- `androidtv.basetv.basetv_async`, [22](#)
- `androidtv.basetv.basetv_sync`, [29](#)
- `androidtv.constants`, [43](#)
- `androidtv.exceptions`, [45](#)
- `androidtv.firetv`, [43](#)
- `androidtv.firetv.base_firetv`, [37](#)
- `androidtv.firetv.firetv_async`, [39](#)
- `androidtv.firetv.firetv_sync`, [41](#)
- `androidtv.setup_async`, [45](#)



## Symbols

<code>_acquire()</code> (in module <code>androidtv.adb_manager.adb_manager_async</code> ), 8	<code>_key()</code> ( <code>androidtv.basetv.basetv_async.BaseTVAsync</code> method), 23
<code>_acquire()</code> (in module <code>androidtv.adb_manager.adb_manager_sync</code> ), 10	<code>_key()</code> ( <code>androidtv.basetv.basetv_sync.BaseTVSync</code> method), 31
<code>_audio_output_device()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 18	<code>_media_session_state()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20
<code>_audio_state()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 18	<code>_parse_device_properties()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> method), 20
<code>_conditions_are_true()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 18	<code>_parse_getevent_line()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20
<code>_current_app()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 19	<code>_parse_stream_music()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20
<code>_current_app_media_session_state()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> method), 19	<code>_running_apps()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20
<code>_custom_state_detection()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> method), 19	<code>_send_intent()</code> ( <code>androidtv.basetv.basetv_async.BaseTVAsync</code> method), 23
<code>_get_hdmi_input()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 19	<code>_send_intent()</code> ( <code>androidtv.basetv.basetv_sync.BaseTVSync</code> method), 31
<code>_get_properties()</code> ( <code>androidtv.androidtv.base_androidtv.BaseAndroidTV</code> method), 15	<code>_update()</code> ( <code>androidtv.androidtv.base_androidtv.BaseAndroidTV</code> method), 16
<code>_get_properties()</code> ( <code>androidtv.firetv.base_firetv.BaseFireTV</code> method), 38	<code>_update()</code> ( <code>androidtv.firetv.base_firetv.BaseFireTV</code> method), 38
<code>_get_stream_music()</code> ( <code>androidtv.basetv.basetv_async.BaseTVAsync</code> method), 23	<code>_volume()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> method), 21
<code>_get_stream_music()</code> ( <code>androidtv.basetv.basetv_sync.BaseTVSync</code> method), 30	<code>_volume_level()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> method), 21
<code>_installed_apps()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 19	<code>_wake_lock_size()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 21
<code>_is_volume_muted()</code> ( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20	
	<b>A</b>
	<code>adb_close()</code> ( <code>androidtv.basetv.basetv_async.BaseTVAsync</code> method), 23
	<code>adb_close()</code> ( <code>androidtv.basetv.basetv_sync.BaseTVSync</code> method), 31
	<code>adb_connect()</code> ( <code>androidtv.basetv.basetv_async.BaseTVAsync</code> method), 23

method), 23

adb\_connect() (androidtv.basetv.basetv\_sync.BaseTVSync method), 31

adb\_pull() (androidtv.basetv.basetv\_async.BaseTVAsync method), 24

adb\_pull() (androidtv.basetv.basetv\_sync.BaseTVSync method), 31

adb\_push() (androidtv.basetv.basetv\_async.BaseTVAsync method), 24

adb\_push() (androidtv.basetv.basetv\_sync.BaseTVSync method), 32

adb\_screenshot() (androidtv.basetv.basetv\_async.BaseTVAsync method), 24

adb\_screenshot() (androidtv.basetv.basetv\_sync.BaseTVSync method), 32

adb\_shell() (androidtv.basetv.basetv\_async.BaseTVAsync method), 24

adb\_shell() (androidtv.basetv.basetv\_sync.BaseTVSync method), 32

ADBPythonAsync (class in androidtv.adb\_manager.adb\_manager\_async), 5

ADBPythonSync (class in androidtv.adb\_manager.adb\_manager\_sync), 8

ADBServerAsync (class in androidtv.adb\_manager.adb\_manager\_async), 6

ADBServerSync (class in androidtv.adb\_manager.adb\_manager\_sync), 9

androidtv (module), 46

androidtv.adb\_manager (module), 11

androidtv.adb\_manager.adb\_manager\_async (module), 5

androidtv.adb\_manager.adb\_manager\_sync (module), 8

androidtv.androidtv (module), 17

androidtv.androidtv.androidtv\_async (module), 11

androidtv.androidtv.androidtv\_sync (module), 13

androidtv.androidtv.base\_androidtv (module), 15

androidtv.basetv (module), 37

androidtv.basetv.basetv (module), 17

androidtv.basetv.basetv\_async (module), 22

androidtv.basetv.basetv\_sync (module), 29

androidtv.constants (module), 43

androidtv.exceptions (module), 45

androidtv.firetv (module), 43

androidtv.firetv.base\_firetv (module), 37

androidtv.firetv.firetv\_async (module), 39

androidtv.firetv.firetv\_sync (module), 41

androidtv.setup\_async (module), 45

AndroidTVAsync (class in androidtv.androidtv.androidtv\_async), 11

AndroidTVSync (class in androidtv.androidtv.androidtv\_sync), 13

audio\_output\_device() (androidtv.basetv.basetv\_async.BaseTVAsync method), 24

audio\_output\_device() (androidtv.basetv.basetv\_sync.BaseTVSync method), 32

audio\_state() (androidtv.basetv.basetv\_async.BaseTVAsync method), 24

audio\_state() (androidtv.basetv.basetv\_sync.BaseTVSync method), 32

available() (androidtv.adb\_manager.adb\_manager\_async.ADBPython property), 5

available() (androidtv.adb\_manager.adb\_manager\_async.ADBServer property), 7

available() (androidtv.adb\_manager.adb\_manager\_sync.ADBPython property), 8

available() (androidtv.adb\_manager.adb\_manager\_sync.ADBServer property), 10

available() (androidtv.basetv.basetv.BaseTV property), 21

awake() (androidtv.basetv.basetv\_async.BaseTVAsync method), 25

awake() (androidtv.basetv.basetv\_sync.BaseTVSync method), 32

## B

back() (androidtv.basetv.basetv\_async.BaseTVAsync method), 25

back() (androidtv.basetv.basetv\_sync.BaseTVSync method), 32

BaseAndroidTV (class in androidtv.androidtv.base\_androidtv), 15

BaseFireTV (class in androidtv.firetv.base\_firetv), 37

BaseTV (class in androidtv.basetv.basetv), 17

BaseTVAsync (class in androidtv.basetv.basetv\_async), 22

BaseTVSync (class in androidtv.basetv.basetv\_sync), 29

## C

ClientAsync (class in androidtv.adb\_manager.adb\_manager\_async), 7



[close\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.ADBPythonAsync](#) (in module [androidtv.constants](#)), [method](#)), [5](#)  
[close\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.ADBServerAsync](#) (in module [androidtv.constants](#)), [method](#)), [7](#)  
[close\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBPythonSync](#) (in module [androidtv.constants](#)), [method](#)), [8](#)  
[close\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBServerSync](#) (in module [androidtv.constants](#)), [method](#)), [10](#)  
[CMD\\_ANDROIDTV\\_PROPERTIES\\_LAZY\\_NO\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [43](#)  
[CMD\\_ANDROIDTV\\_PROPERTIES\\_LAZY\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [43](#)  
[CMD\\_ANDROIDTV\\_PROPERTIES\\_NOT\\_LAZY\\_NO\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [43](#)  
[CMD\\_ANDROIDTV\\_PROPERTIES\\_NOT\\_LAZY\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [43](#)  
[CMD\\_AUDIO\\_STATE](#) (in module [androidtv.constants](#)), [43](#)  
[CMD\\_AWAKE](#) (in module [androidtv.constants](#)), [43](#)  
[CMD\\_CURRENT\\_APP](#) (in module [androidtv.constants](#)), [43](#)  
[CMD\\_CURRENT\\_APP\\_GOOGLE\\_TV](#) (in module [androidtv.constants](#)), [43](#)  
[CMD\\_FIRETV\\_PROPERTIES\\_LAZY\\_NO\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [43](#)  
[CMD\\_FIRETV\\_PROPERTIES\\_LAZY\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [43](#)  
[CMD\\_FIRETV\\_PROPERTIES\\_NOT\\_LAZY\\_NO\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [44](#)  
[CMD\\_FIRETV\\_PROPERTIES\\_NOT\\_LAZY\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [44](#)  
[CMD\\_GOOGLE\\_TV\\_PROPERTIES\\_LAZY\\_NO\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [44](#)  
[CMD\\_GOOGLE\\_TV\\_PROPERTIES\\_LAZY\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [44](#)  
[CMD\\_GOOGLE\\_TV\\_PROPERTIES\\_NOT\\_LAZY\\_NO\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [44](#)  
[CMD\\_GOOGLE\\_TV\\_PROPERTIES\\_NOT\\_LAZY\\_RUNNING\\_APPS](#) ([in module androidtv.constants](#)), [44](#)  
[CMD\\_HDMI\\_INPUT](#) (in module [androidtv.constants](#)), [44](#)  
[CMD\\_INSTALLED\\_APPS](#) (in module [androidtv.constants](#)), [44](#)  
[CMD\\_LAUNCH\\_APP](#) (in module [androidtv.constants](#)), [44](#)  
[CMD\\_LAUNCH\\_APP\\_GOOGLE\\_TV](#) (in module [androidtv.constants](#)), [44](#)  
[CMD\\_MEDIA\\_SESSION\\_STATE](#) (in module [androidtv.constants](#)), [44](#)  
[CMD\\_MEDIA\\_SESSION\\_STATE\\_FULL](#) (in module [androidtv.constants](#)), [44](#)  
[CMD\\_RUNNING\\_APPS\\_ANDROIDTV](#) (in module [androidtv.constants](#)), [44](#)  
[CMD\\_RUNNING\\_APPS\\_FIRETV](#) (in module [androidtv.constants](#)), [44](#)  
[CMD\\_SCREEN\\_ON](#) (in module [androidtv.constants](#)), [44](#)

**D**  
[DEFAULT\\_ADB\\_TIMEOUT\\_S](#) (in module [androidtv.constants](#)), [45](#)  
[DEFAULT\\_AUTH\\_TIMEOUT\\_S](#) (in module [androidtv.constants](#)), [45](#)  
[DEFAULT\\_LOCK\\_TIMEOUT\\_S](#) (in module [androidtv.constants](#)), [45](#)  
[device\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.ClientAsync](#) (class in [androidtv.adb\\_manager.adb\\_manager\\_async](#)), [method](#)), [7](#)  
[DEVICE\\_CLASS](#) ([androidtv.androidtv.base\\_androidtv.BaseAndroidTV](#) attribute), [15](#)  
[DEVICE\\_CLASS](#) ([androidtv.firetv.base\\_firetv.BaseFireTV](#) attribute), [38](#)  
[DeviceAsync](#) (class in [androidtv.adb\\_manager.adb\\_manager\\_async](#)), [7](#)  
[DeviceSync](#) (class in [androidtv.adb\\_manager.adb\\_manager\\_sync](#)), [7](#)  
[down\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) (class in [androidtv.basetv.basetv\\_async](#)), [method](#)), [25](#)  
[down\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) (class in [androidtv.basetv.basetv\\_sync](#)), [method](#)), [33](#)

**E**  
[enter\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) (class in [androidtv.basetv.basetv\\_async](#)), [method](#)), [25](#)  
[enter\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) (class in [androidtv.basetv.basetv\\_sync](#)), [method](#)), [33](#)

**F**  
[FireTVAsync](#) (class in [androidtv.firetv.firetv\\_async](#)), [39](#)  
[FireTVSync](#) (class in [androidtv.firetv.firetv\\_sync](#)), [41](#)

**G**  
[get\\_device\\_properties\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) (class in [androidtv.basetv.basetv\\_async](#)), [method](#)), [25](#)  
[get\\_device\\_properties\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) (class in [androidtv.basetv.basetv\\_sync](#)), [method](#)), [33](#)

<i>method</i> ), 25		<i>is_volume_muted()</i> (an-	
<i>get_device_properties()</i> (an-	<i>droidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33	<i>droidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33	
<i>get_hdmi_input()</i> (an-	<i>droidtv.basetv.basetv_async.BaseTVAsync method</i> ), 25	<b>K</b>	
<i>get_hdmi_input()</i> (an-	<i>droidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33	<i>key_0()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 25
<i>get_installed_apps()</i> (an-	<i>droidtv.basetv.basetv_async.BaseTVAsync method</i> ), 25	<i>key_0()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33
<i>get_installed_apps()</i> (an-	<i>droidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33	<i>key_1()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 25
<i>get_properties()</i> (an-	<i>droidtv.androidtv.androidtv_async.AndroidTVAsync method</i> ), 11	<i>key_1()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33
<i>get_properties()</i> (an-	<i>droidtv.androidtv.androidtv_sync.AndroidTVSync method</i> ), 13	<i>key_2()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
<i>get_properties()</i> (an-	<i>droidtv.firetv.firetv_async.FireTVAsync method</i> ), 39	<i>key_2()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33
<i>get_properties()</i> (an-	<i>droidtv.firetv.firetv_sync.FireTVSync method</i> ), 41	<i>key_3()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
<i>get_properties_dict()</i> (an-	<i>droidtv.androidtv.androidtv_async.AndroidTVAsync method</i> ), 12	<i>key_3()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33
<i>get_properties_dict()</i> (an-	<i>droidtv.androidtv.androidtv_sync.AndroidTVSync method</i> ), 14	<i>key_4()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
<i>get_properties_dict()</i> (an-	<i>droidtv.firetv.firetv_async.FireTVAsync method</i> ), 40	<i>key_4()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33
<i>get_properties_dict()</i> (an-	<i>droidtv.firetv.firetv_sync.FireTVSync method</i> ), 42	<i>key_5()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
<b>H</b>		<i>key_5()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33
<i>ha_state_detection_rules_validator()</i> (in	<i>module androidtv</i> ), 46	<i>key_6()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
<i>home()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 25	<i>key_6()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33
<i>home()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33	<i>key_7()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
<b>I</b>		<i>key_7()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 33
<i>is_volume_muted()</i> (an-	<i>droidtv.basetv.basetv_async.BaseTVAsync method</i> ), 25	<i>key_8()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
		<i>key_8()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 34
		<i>key_9()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
		<i>key_9()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 34
		<i>key_a()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
		<i>key_a()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 34
		<i>key_b()</i> (an-	<i>androidtv.basetv.basetv_async.BaseTVAsync method</i> ), 26
		<i>key_b()</i> (an-	<i>androidtv.basetv.basetv_sync.BaseTVSync method</i> ), 34

[illegible]

## L

```

launch_app () (androidtv.basetv.basetv_async.BaseTVAsync
               method), 27
launch_app () (androidtv.basetv.basetv_sync.BaseTVSync
               method), 35
learn_sendevent () (an-
                   droidtv.basetv.basetv_async.BaseTVAsync
                   method), 27
learn_sendevent () (an-
                   droidtv.basetv.basetv_sync.BaseTVSync
                   method), 35

```

[left\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 27  
[left\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 35  
[load\\_adbkey\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.ADBPythonAsync](#) static method), 6  
[load\\_adbkey\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBPythonSync](#) static method), 9  
[LOCK\\_KWARGS](#) (in module [androidtv.adb\\_manager.adb\\_manager\\_sync](#)), 10  
[LockNotAcquiredException](#), 45

## M

[media\\_next\\_track\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 27  
[media\\_next\\_track\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 35  
[media\\_pause\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[media\\_pause\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 35  
[media\\_play\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[media\\_play\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 35  
[media\\_play\\_pause\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[media\\_play\\_pause\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 35  
[media\\_previous\\_track\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[media\\_previous\\_track\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 35  
[media\\_session\\_state\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[media\\_session\\_state\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 35  
[MEDIA\\_SESSION\\_STATES](#) (in module [androidtv.constants](#)), 45  
[media\\_stop\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28

[media\\_stop\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 36  
[menu\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[menu\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 36  
[mute\\_volume\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[mute\\_volume\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 36

## P

[power\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[power\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 36  
[pull\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.ADBPythonAsync](#) method), 6  
[pull\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.ADBServerAsync](#) method), 7  
[pull\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.DeviceAsync](#) method), 8  
[pull\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBPythonSync](#) method), 9  
[pull\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBServerSync](#) method), 10  
[push\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.ADBPythonAsync](#) method), 6  
[push\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.ADBServerAsync](#) method), 7  
[push\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.DeviceAsync](#) method), 8  
[push\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBPythonSync](#) method), 9  
[push\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBServerSync](#) method), 10

## R

[right\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 28  
[right\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 36  
[running\\_apps\(\)](#) ([androidtv.androidtv.androidtv\\_async.AndroidTVAsync](#) method), 12  
[running\\_apps\(\)](#) ([androidtv.androidtv.androidtv\\_sync.AndroidTVSync](#) method), 14  
[running\\_apps\(\)](#) ([androidtv.firetv.firetv\\_async.FireTVAsync](#) method), 40

running\_apps() (androidtv.firetv.firetv\_sync.FireTVSync method), 42

**S**

screen\_on() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

screen\_on() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

screencap() (androidtv.adb\_manager.adb\_manager\_async.ADBPythonAsync method), 6

screencap() (androidtv.adb\_manager.adb\_manager\_async.ADBServerAsync method), 7

screencap() (androidtv.adb\_manager.adb\_manager\_async.DeviceAsync method), 8

screencap() (androidtv.adb\_manager.adb\_manager\_sync.ADBPythonSync method), 9

screencap() (androidtv.adb\_manager.adb\_manager\_sync.ADBServerSync method), 10

set\_volume\_level() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

set\_volume\_level() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

setup() (in module androidtv), 46

setup() (in module androidtv.setup\_async), 45

shell() (androidtv.adb\_manager.adb\_manager\_async.ADBPythonAsync method), 6

shell() (androidtv.adb\_manager.adb\_manager\_async.ADBServerAsync method), 7

shell() (androidtv.adb\_manager.adb\_manager\_async.DeviceAsync method), 8

shell() (androidtv.adb\_manager.adb\_manager\_sync.ADBPythonSync method), 9

shell() (androidtv.adb\_manager.adb\_manager\_sync.ADBServerSync method), 10

sleep() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

sleep() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

space() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

space() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

start\_intent() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

start\_intent() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

state\_detection\_rules\_validator() (in module androidtv.basetv.basetv), 21

stop\_app() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

stop\_app() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

**T**

turn\_off() (androidtv.androidtv.androidtv\_async.AndroidTVAsync method), 12

turn\_off() (androidtv.androidtv.androidtv\_sync.AndroidTVSync method), 14

turn\_off() (androidtv.firetv.firetv\_async.FireTVAsync method), 40

turn\_off() (androidtv.firetv.firetv\_sync.FireTVSync method), 42

turn\_on() (androidtv.androidtv.androidtv\_async.AndroidTVAsync method), 12

turn\_on() (androidtv.androidtv.androidtv\_sync.AndroidTVSync method), 15

turn\_on() (androidtv.firetv.firetv\_async.FireTVAsync method), 40

turn\_on() (androidtv.firetv.firetv\_sync.FireTVSync method), 42

**U**

up() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

up() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

update() (androidtv.androidtv.androidtv\_async.AndroidTVAsync method), 12

update() (androidtv.androidtv.androidtv\_sync.AndroidTVSync method), 15

update() (androidtv.firetv.firetv\_async.FireTVAsync method), 40

update() (androidtv.firetv.firetv\_sync.FireTVSync method), 42

**V**

VALID\_PROPERTIES (in module androidtv.constants), 45

VALID\_PROPERTIES\_TYPES (in module androidtv.constants), 45

VALID\_STATE\_PROPERTIES (in module androidtv.constants), 45

VALID\_STATES (in module androidtv.constants), 45

volume() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

volume() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

volume\_down() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

`volume_down()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), [37](#)

`volume_level()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), [29](#)

`volume_level()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), [37](#)

`volume_up()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), [29](#)

`volume_up()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), [37](#)

## W

`wake_lock_size()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), [29](#)

`wake_lock_size()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), [37](#)