

---

# **androidtv Documentation**

***Release 0.0.64***

**Jeff Irion**

**Feb 15, 2022**



# 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*, *transport\_timeout\_s=1.0*)

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)
- **transport\_timeout\_s** (*float*) – Transport 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 screenshot** ()

Take a screenshot using the Python ADB implementation.

**Returns** The screenshot 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 screencap()**

Take a screenshot using an ADB server.

**Returns** The screencap 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.AdbDeviceUsbAsync (serial=None,
                                                                port_path=None,
                                                                de-
                                                                fault_transport_timeout_s=None,
                                                                ban-
                                                                ner=None)
```

Bases: `object`

An async wrapper for the adb-shell `AdbDeviceUsb` class.

**property available**

Whether or not an ADB connection to the device has been established.

**async close()**

Close the connection via the provided transport's `close()` method.

**async connect** (*rsa\_keys=None, transport\_timeout\_s=None, auth\_timeout\_s=10.0, read\_timeout\_s=10.0, auth\_callback=None*)

Establish an ADB connection to the device.

**async pull** (*device\_path, local\_path, progress\_callback=None, transport\_timeout\_s=None, read\_timeout\_s=10.0*)

Pull a file from the device.

**async push** (*local\_path, device\_path, st\_mode=33272, mtime=0, progress\_callback=None, transport\_timeout\_s=None, read\_timeout\_s=10.0*)

Push a file or directory to the device.

**async shell** (*command, transport\_timeout\_s=None, read\_timeout\_s=10.0, timeout\_s=None, decode=True*)

Send an ADB shell command to the device.

**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.

- *ADBPpythonSync* 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.**ADBPpythonSync** (*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 *ADBPpythonSync.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, transport\_timeout\_s=1.0*)

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)
- **transport\_timeout\_s** (*float*) – Transport 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

**screenshot** ()

Take a screenshot using the Python ADB implementation.

**Returns** The screenshot 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()`

**classmethod from\_base** (*base\_tv*)

Construct an *AndroidTVAsync* object from a *BaseTVAsync* object.

**Parameters** **base\_tv** (*BaseTVAsync*) – The object that will be converted to an *AndroidTVAsync* object

**Returns** **atv** – The constructed *AndroidTVAsync* object

**Return type** *AndroidTVAsync*

**async get\_properties** (*get\_running\_apps=True*, *lazy=False*)

Get the properties needed for Home Assistant updates.

#### 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 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()`

**classmethod from\_base** (*base\_tv*)

Construct an *AndroidTVSync* object from a *BaseTVSync* object.

**Parameters** **base\_tv** ([BaseTVSync](#)) – The object that will be converted to an *AndroidTVSync* object

**Returns** **atv** – The constructed *AndroidTVSync* object

**Return type** [AndroidTVSync](#)

**get\_properties** (*get\_running\_apps=True*, *lazy=False*)

Get the properties needed for Home Assistant updates.

**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

**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'

**DEVICE\_ENUM** = 1

**\_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)

**DEVICE\_ENUM** = 0

**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*

**\_cmd\_current\_app** ()

Get the command used to retrieve the current app for this device.

**Returns** The device-specific ADB shell command used to determine the current app

**Return type** *str*

**\_cmd\_current\_app\_media\_session\_state** ()

Get the command used to retrieve the current app and media session state for this device.

**Returns** The device-specific ADB shell command used to determine the current app and media session state

**Return type** *str*

**\_cmd\_launch\_app** (*app*)

Get the command to launch the specified app for this device.

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

**Returns** The device-specific command to launch the app

**Return type** *str*

**\_cmd\_running\_apps** ()

Get the command used to retrieve the running apps for this device.

**Returns** The device-specific ADB shell command used to determine the running apps

**Return type** *str*

**\_cmd\_turn\_off** ()

Get the command used to turn off this device.

**Returns** The device-specific ADB shell command used to turn off the device

**Return type** *str*

**`_cmd_turn_on()`**

Get the command used to turn on this device.

**Returns** The device-specific ADB shell command used to turn on the device

**Return type** `str`

**`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.basetv.basetv.BaseTV._cmd_current_app`.

**Parameters** **`current_app_response`** (*str*, *None*) – The output from the ADB command `androidtv.basetv.basetv.BaseTV._cmd_current_app`

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

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

**`_current_app_media_session_state`** (*current\_app\_media\_session\_state\_response*)

Get the current app and the media session state properties from the output of `androidtv.basetv.basetv.BaseTV._cmd_current_app_media_session_state`.

**Parameters** **`current_app_media_session_state_response`** (*str*, *None*) – The output of `androidtv.basetv.basetv.BaseTV._cmd_current_app_media_session_state`

**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** `_get_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`

**\_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
- **method fills in the device\_properties attribute, which is a dictionary with keys (This) –**
- **'manufacturer', 'model', and 'sw\_version' ('serialno',) –**

**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_mac_address (mac_response)`

Parse a MAC address from the ADB shell response.

**Parameters** `mac_response (str, None)` – The response from the MAC address ADB shell command

**Returns** The parsed MAC address, or `None` if it could not be determined

**Return type** `str, None`



**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** `_remove_adb_shell_prefix` (*cmd*)

Remove the ‘adb shell ‘ prefix from *cmd*, if present.

**Parameters** `cmd` (*str*) – The ADB shell command

**Returns** *cmd* with the ‘adb shell ‘ prefix removed, if it was present

**Return type** *str*

**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*

**static** `_screen_on_awake_wake_lock_size` (*output*)

Check if the screen is on and the device is awake, and get the wake lock size.

**Parameters** `output` (*str*, *None*) – The output from `androidtv.constants.CMD_SCREEN_ON_AWAKE_WAKE_LOCK_SIZE`

**Returns**

- *bool*, *None* – Whether or not the device is on, or *None* if it could not be determined
- *bool*, *None* – Whether or not the device is awake (screensaver is not running), or *None* if it could not be determined
- *int*, *None* – The size of the current wake lock, or *None* if it could not be determined

**\_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`

**customize\_command** (*custom\_command*, *value*)

Customize a command used to retrieve properties.

**Parameters**

- **custom\_command** (*str*) – The name of the command that will be customized; it must be in `constants.CUSTOMIZABLE_COMMANDS`
- **value** (*str*, *None*) – The custom ADB command that will be used, or *None* if the custom command should be deleted

`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, 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_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*, *transport\_timeout\_s=1.0*)

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)
- **transport\_timeout\_s** (*float*) – Transport 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 current\_app\_media\_session\_state()**

Get the current app and the state from the output of `dumpsys media_session`.

**Returns**

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

**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 running\_apps()**

Return a list of running user applications.

**Returns** A list of the running apps

**Return type** `list`

**async screen\_on()**

Check if the screen is on.

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

**Return type** `bool`

**async screen\_on\_awake\_wake\_lock\_size()**

Check if the screen is on and the device is awake, and get the wake lock size.

**Returns**

- *bool* – Whether or not the device is on
- *bool* – Whether or not the device is awake (screensaver is not running)
- *int*, *None* – The size of the current wake lock, or `None` if it could not be determined

**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 stream\_music\_properties** ()

Get various properties from the “STREAM\_MUSIC” block from `dumpsys audio..`

**Returns**

- **audio\_output\_device** (*str*, *None*) – The current audio playback device, or `None` if it could not be determined
- **is\_volume\_muted** (*bool*, *None*) – Whether or not the volume is muted, or `None` if it could not be determined
- **volume** (*int*, *None*) – The absolute volume level, or `None` if it could not be determined
- **volume\_level** (*float*, *None*) – The volume level (between 0 and 1), or `None` if it could not be determined

**async turn\_off** ()

Turn off the device.

**async turn\_on** ()

Turn on the device.

**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, adb_server_ip="  

bkey=", 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*, *transport\_timeout\_s=1.0*)

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)
- **transport\_timeout\_s** (*float*) – Transport 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

**current\_app\_media\_session\_state** ()

Get the current app and the state from the output of `dumpsys media_session`.

**Returns**

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

**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.

**running\_apps()**  
Return a list of running user applications.

**Returns** A list of the running apps

**Return type** list

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

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

**Return type** bool

**screen\_on\_awake\_wake\_lock\_size()**  
Check if the screen is on and the device is awake, and get the wake lock size.

**Returns**

- *bool* – Whether or not the device is on
- *bool* – Whether or not the device is awake (screensaver is not running)
- *int, None* – The size of the current wake lock, or *None* if it could not be determined

**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*

**stream\_music\_properties()**  
Get various properties from the “STREAM\_MUSIC” block from `dumpsys audio..`

**Returns**

- **audio\_output\_device** (*str, None*) – The current audio playback device, or *None* if it could not be determined
- **is\_volume\_muted** (*bool, None*) – Whether or not the volume is muted, or *None* if it could not be determined
- **volume** (*int, None*) – The absolute volume level, or *None* if it could not be determined
- **volume\_level** (*float, None*) – The volume level (between 0 and 1), or *None* if it could not be determined

**turn\_off** ()

Turn off the device.

**turn\_on** ()

Turn on the device.

**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, adbkey="", 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'
```

```
DEVICE_ENUM = 2
```

```
_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()`

**classmethod from\_base** (*base\_tv*)

Construct a `FireTVAsync` object from a `BaseTVAsync` object.

**Parameters** **base\_tv** (`BaseTVAsync`) – The object that will be converted to a `FireTVAsync` object

**Returns** **ftv** – The constructed `FireTVAsync` object

**Return type** `FireTVAsync`

**async get\_properties** (*get\_running\_apps=True, lazy=False*)

Get the properties needed for Home Assistant updates.

### 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 `dumpps 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 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_server_ip=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\(\)](#)

**classmethod from\_base** (*base\_tv*)

Construct a *FireTVSync* object from a *BaseTVSync* object.

**Parameters** **base\_tv** ([BaseTVSync](#)) – The object that will be converted to a *FireTVSync* object

**Returns** **ftv** – The constructed *FireTVSync* object

**Return type** [FireTVSync](#)

**get\_properties** (*get\_running\_apps=True*, *lazy=False*)

Get the properties needed for Home Assistant updates.

#### 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

**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_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 -E 'mCurrentApp')"
Output identifier for current/focused application
```

```
androidtv.constants.CMD_CURRENT_APP_GOOGLE_TV = 'CURRENT_APP=$(dumpsys activity a . | grep -E 'mCurrentApp')'
Output identifier for current/focused application (for a Google TV device)
```



```

androidtv.constants.CMD_CURRENT_APP_MEDIA_SESSION_STATE = "CURRENT_APP=$(dumpsys window wi
    Determine the current app and get the state from dumpsys media_session

androidtv.constants.CMD_CURRENT_APP_MEDIA_SESSION_STATE_GOOGLE_TV = "CURRENT_APP=$(dumpsys
    Determine the current app and get the state from dumpsys media_session for a Google TV device

androidtv.constants.CMD_DEFINE_CURRENT_APP_VARIABLE = "CURRENT_APP=$(dumpsys window windows
    Assign focused application identifier to CURRENT_APP variable

androidtv.constants.CMD_DEFINE_CURRENT_APP_VARIABLE_GOOGLE_TV = 'CURRENT_APP=$(dumpsys acti
    Assign focused application identifier to CURRENT_APP variable (for a Google TV device)

androidtv.constants.CMD_DEVICE_PROPERTIES = 'getprop ro.product.manufacturer && getprop ro
    The command used for getting the device properties

androidtv.constants.CMD_HDMI_INPUT = "dumpsys activity starter | grep -E -o '(ExternalTv|HI
    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 -E 'mCur
    Launch an app if it is not already the current app

androidtv.constants.CMD_LAUNCH_APP_CONDITION = "if [ $CURRENT_APP != '{0}' ]; then monkey -
    Launch an app if it is not already the current app (assumes the variable CURRENT_APP has already been set)

androidtv.constants.CMD_LAUNCH_APP_GOOGLE_TV = "CURRENT_APP=$(dumpsys activity a . | grep m
    Launch an app on a Google TV device

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_PARSE_CURRENT_APP = 'CURRENT_APP=${CURRENT_APP##*ActivityRecord{* *
    Parse current application identifier from dumpsys output and assign it to CURRENT_APP variable (assumes
    dumpsys output is momentarily set to CURRENT_APP variable)

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_SCREEN_ON_AWAKE_WAKE_LOCK_SIZE = "(dumpsys power | grep 'Display Po
    Determine if the device is on, the screen is on, and get the wake lock size

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

androidtv.constants.CMD_TURN_OFF_ANDROIDTV = "(dumpsys power | grep 'Display Power' | grep
    KEY_POWER = 26 is defined below)

    Type Turn off an Android TV device (note

androidtv.constants.CMD_TURN_OFF_FIRETV = "(dumpsys power | grep 'Display Power' | grep -q
    KEY_SLEEP = 223 is defined below)

    Type Turn off a Fire TV device (note

```

```
androidtv.constants.CMD_TURN_ON_ANDROIDTV = "(dumpsys power | grep 'Display Power' | grep -q KEY_POWER = 26 is defined below)"
```

**Type** Turn on an Android TV device (note

```
androidtv.constants.CMD_TURN_ON_FIRETV = "(dumpsys power | grep 'Display Power' | grep -q KEY_POWER = 26 and KEY_HOME = 3 are defined below)"
```

**Type** Turn on a Fire TV device (note

```
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.DEFAULT_TRANSPORT_TIMEOUT_S = 1.0
```

Default transport timeout (in s) for adb\_shell.handle.tcp\_handle.TcpHandle.connect() and adb\_shell.handle.tcp\_handle\_async.TcpHandleAsync.connect()

**class** androidtv.constants.DeviceEnum

Bases: enum.IntEnum

An enum for the various device types.

**ANDROIDTV** = 1

**BASETV** = 0

**FIRETV** = 2

```
androidtv.constants.HA_CUSTOMIZABLE_COMMANDS = ('current_app_media_session_state', 'launch')
```

The subset of *CUSTOMIZABLE\_COMMANDS* that is potentially used in the `update()` method

```
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, *transport\_timeout\_s*=1.0)

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()`
- **transport\_timeout\_s** (*float*) – Transport timeout (in seconds)

**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, transport_timeout_s=1.0)
```

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()`
- **transport\_timeout\_s** (*float*) – Transport timeout (in seconds)

**Returns** The representation of the device

**Return type** *AndroidTVSync*, *FireTVSync*

## 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`, [47](#)
- `androidtv.adb_manager`, [11](#)
- `androidtv.adb_manager.adb_manager_async`,  
[5](#)
- `androidtv.adb_manager.adb_manager_sync`,  
[9](#)
- `androidtv.androidtv`, [16](#)
- `androidtv.androidtv.androidtv_async`, [11](#)
- `androidtv.androidtv.androidtv_sync`, [13](#)
- `androidtv.androidtv.base_androidtv`, [15](#)
- `androidtv.basetv`, [40](#)
- `androidtv.basetv.basetv`, [17](#)
- `androidtv.basetv.basetv_async`, [23](#)
- `androidtv.basetv.basetv_sync`, [31](#)
- `androidtv.constants`, [44](#)
- `androidtv.exceptions`, [46](#)
- `androidtv.firetv`, [44](#)
- `androidtv.firetv.base_firetv`, [40](#)
- `androidtv.firetv.firetv_async`, [41](#)
- `androidtv.firetv.firetv_sync`, [42](#)
- `androidtv.setup_async`, [47](#)



## Symbols

<code>_acquire()</code>	(in module <code>androidtv.adb_manager.adb_manager_async</code> ), 8	<code>_get_stream_music()</code>	( <code>androidtv.basetv.basetv_sync.BaseTVSync</code> method), 32
<code>_acquire()</code>	(in module <code>androidtv.adb_manager.adb_manager_sync</code> ), 11	<code>_is_volume_muted()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20
<code>_audio_output_device()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 18	<code>_key()</code>	( <code>androidtv.basetv.basetv_async.BaseTVAsync</code> method), 24
<code>_audio_state()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 18	<code>_key()</code>	( <code>androidtv.basetv.basetv_sync.BaseTVSync</code> method), 32
<code>_cmd_current_app()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 18	<code>_parse_device_properties()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 20
<code>_cmd_current_app_media_session_state()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 18	<code>_parse_getevent_line()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20
<code>_cmd_launch_app()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 18	<code>_parse_mac_address()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20
<code>_cmd_running_apps()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 18	<code>_parse_stream_music()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20
<code>_cmd_turn_off()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 18	<code>_remove_adb_shell_prefix()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 21
<code>_cmd_turn_on()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 18	<code>_running_apps()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 21
<code>_conditions_are_true()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 19	<code>_screen_on_awake_wake_lock_size()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 21
<code>_current_app()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 19	<code>_send_intent()</code>	( <code>androidtv.basetv.basetv_async.BaseTVAsync</code> method), 24
<code>_current_app_media_session_state()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 19	<code>_send_intent()</code>	( <code>androidtv.basetv.basetv_sync.BaseTVSync</code> method), 32
<code>_custom_state_detection()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 19	<code>_update()</code>	( <code>androidtv.androidtv.base_androidtv.BaseAndroidTV</code> method), 16
<code>_get_hdmi_input()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20	<code>_update()</code>	( <code>androidtv.firetv.base_firetv.BaseFireTV</code> method), 40
<code>_get_installed_apps()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> static method), 20	<code>_volume()</code>	( <code>androidtv.basetv.basetv.BaseTV</code> method), 21
<code>_get_stream_music()</code>	( <code>androidtv.basetv.basetv_async.BaseTVAsync</code> method), 24	<code>_volume_level()</code>	( <code>androidtv.basetv.basetv.BaseTV</code>

*method*), 21

`_wake_lock_size()` (*androidtv.basetv.basetv.BaseTV static method*), 21

## A

`adb_close()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 24

`adb_close()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 33

`adb_connect()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 24

`adb_connect()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 33

`adb_pull()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 24

`adb_pull()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 33

`adb_push()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 25

`adb_push()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 33

`adb_screenshot()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 25

`adb_screenshot()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 33

`adb_shell()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 25

`adb_shell()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 34

`AdbDeviceUsbAsync` (*class in androidtv.adb\_manager.adb\_manager\_async*), 7

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

`ADBPythonSync` (*class in androidtv.adb\_manager.adb\_manager\_sync*), 9

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

`ADBServerSync` (*class in androidtv.adb\_manager.adb\_manager\_sync*), 10

`ANDROIDTV` (*androidtv.constants.DeviceEnum attribute*), 46

`androidtv` (*module*), 47

`androidtv.adb_manager` (*module*), 11

`androidtv.adb_manager.adb_manager_async` (*module*), 5

`androidtv.adb_manager.adb_manager_sync` (*module*), 9

`androidtv.androidtv` (*module*), 16

`androidtv.androidtv.androidtv_async` (*module*), 11

`androidtv.androidtv.androidtv_sync` (*module*), 13

`androidtv.androidtv.base_androidtv` (*module*), 15

`androidtv.basetv` (*module*), 40

`androidtv.basetv.basetv` (*module*), 17

`androidtv.basetv.basetv_async` (*module*), 23

`androidtv.basetv.basetv_sync` (*module*), 31

`androidtv.constants` (*module*), 44

`androidtv.exceptions` (*module*), 46

`androidtv.firetv` (*module*), 44

`androidtv.firetv.base_firetv` (*module*), 40

`androidtv.firetv.firetv_async` (*module*), 41

`androidtv.firetv.firetv_sync` (*module*), 42

`androidtv.setup_async` (*module*), 47

`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*), 25

`audio_output_device()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 34

`audio_state()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 25

`audio_state()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 34

`available()` (*androidtv.adb\_manager.adb\_manager\_async.AdbDeviceUsbAsync property*), 8

`available()` (*androidtv.adb\_manager.adb\_manager\_async.ADBPythonAsync property*), 5

`available()` (*androidtv.adb\_manager.adb\_manager\_async.ADBServerAsync property*), 7

`available()` (*androidtv.adb\_manager.adb\_manager\_sync.ADBPythonSync property*), 9

`available()` (*androidtv.adb\_manager.adb\_manager\_sync.ADBServerSync property*), 10

`available()` (*androidtv.basetv.basetv.BaseTV property*), 22

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

`awake()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 34



## B

`back()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 26

`back()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 34

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

`BaseFireTV` (class in *androidtv.firetv.base\_firetv*), 40

`BASETV` (*androidtv.constants.DeviceEnum attribute*), 46

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

`BaseTVAsync` (class in *androidtv.basetv.basetv\_async*), 23

`BaseTVSync` (class in *androidtv.basetv.basetv\_sync*), 31

## C

`ClientAsync` (class in *androidtv.adb\_manager.adb\_manager\_async*), 8

`close()` (*androidtv.adb\_manager.adb\_manager\_async.AdbDeviceUsbAsync method*), 8

`close()` (*androidtv.adb\_manager.adb\_manager\_async.ADBPythonAsync method*), 5

`close()` (*androidtv.adb\_manager.adb\_manager\_async.ADBServerAsync method*), 7

`close()` (*androidtv.adb\_manager.adb\_manager\_sync.ADBPythonSync method*), 9

`close()` (*androidtv.adb\_manager.adb\_manager\_sync.ADBServerSync method*), 10

`CMD_AUDIO_STATE` (in module *androidtv.constants*), 44

`CMD_AWAKE` (in module *androidtv.constants*), 44

`CMD_CURRENT_APP` (in module *androidtv.constants*), 44

`CMD_CURRENT_APP_GOOGLE_TV` (in module *androidtv.constants*), 44

`CMD_CURRENT_APP_MEDIA_SESSION_STATE` (in module *androidtv.constants*), 44

`CMD_CURRENT_APP_MEDIA_SESSION_STATE_GOOGLE_TV` (in module *androidtv.constants*), 45

`CMD_DEFINE_CURRENT_APP_VARIABLE` (in module *androidtv.constants*), 45

`CMD_DEFINE_CURRENT_APP_VARIABLE_GOOGLE_TV` (in module *androidtv.constants*), 45

`CMD_DEVICE_PROPERTIES` (in module *androidtv.constants*), 45

`CMD_HDMI_INPUT` (in module *androidtv.constants*), 45

`CMD_INSTALLED_APPS` (in module *androidtv.constants*), 45

`CMD_LAUNCH_APP` (in module *androidtv.constants*), 45

`CMD_LAUNCH_APP_CONDITION` (in module *androidtv.constants*), 45

`CMD_LAUNCH_APP_GOOGLE_TV` (in module *androidtv.constants*), 45

`CMD_MEDIA_SESSION_STATE` (in module *androidtv.constants*), 45

`CMD_PARSE_CURRENT_APP` (in module *androidtv.constants*), 45

`CMD_RUNNING_APPS_ANDROIDTV` (in module *androidtv.constants*), 45

`CMD_RUNNING_APPS_FIRETV` (in module *androidtv.constants*), 45

`CMD_SCREEN_ON` (in module *androidtv.constants*), 45

`CMD_SCREEN_ON_AWAKE_WAKE_LOCK_SIZE` (in module *androidtv.constants*), 45

`CMD_STREAM_MUSIC` (in module *androidtv.constants*), 45

`CMD_TURN_OFF_ANDROIDTV` (in module *androidtv.constants*), 45

`CMD_TURN_OFF_FIRETV` (in module *androidtv.constants*), 45

`CMD_TURN_ON_ANDROIDTV` (in module *androidtv.constants*), 45

`CMD_TURN_ON_FIRETV` (in module *androidtv.constants*), 46

`CMD_WAKE_LOCK_SIZE` (in module *androidtv.constants*), 46

`connect()` (*androidtv.adb\_manager.adb\_manager\_async.AdbDeviceUsbAsync method*), 8

`connect()` (*androidtv.adb\_manager.adb\_manager\_async.ADBPythonAsync method*), 5

`connect()` (*androidtv.adb\_manager.adb\_manager\_async.ADBServerAsync method*), 7

`connect()` (*androidtv.adb\_manager.adb\_manager\_sync.ADBPythonSync method*), 9

`connect()` (*androidtv.adb\_manager.adb\_manager\_sync.ADBServerSync method*), 10

`current_app()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 26

`current_app()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 34

`current_app_media_session_state()` (*androidtv.basetv.basetv\_async.BaseTVAsync method*), 26

`current_app_media_session_state()` (*androidtv.basetv.basetv\_sync.BaseTVSync method*), 34

`customize_command()` (*androidtv.basetv.basetv.BaseTV method*), 22

## D

`DEFAULT_ADB_TIMEOUT_S` (in module *androidtv.constants*), 46

`DEFAULT_AUTH_TIMEOUT_S` (in module *androidtv.constants*), 46

[DEFAULT\\_LOCK\\_TIMEOUT\\_S](#) (in module `androidtv.constants`), 46  
[DEFAULT\\_TRANSPORT\\_TIMEOUT\\_S](#) (in module `androidtv.constants`), 46  
[device\(\)](#) (`androidtv.adb_manager.adb_manager_async.ClientAsync` method), 8  
[DEVICE\\_CLASS](#) (`androidtv.androidtv.base_androidtv.BaseAndroidTV` attribute), 16  
[DEVICE\\_CLASS](#) (`androidtv.firetv.base_firetv.BaseFireTV` attribute), 40  
[DEVICE\\_ENUM](#) (`androidtv.androidtv.base_androidtv.BaseAndroidTV` attribute), 16  
[DEVICE\\_ENUM](#) (`androidtv.basetv.basetv.BaseTV` attribute), 18  
[DEVICE\\_ENUM](#) (`androidtv.firetv.base_firetv.BaseFireTV` attribute), 40  
[DeviceAsync](#) (class in `androidtv.adb_manager.adb_manager_async`), 8  
[DeviceEnum](#) (class in `androidtv.constants`), 46  
[down\(\)](#) (`androidtv.basetv.basetv_async.BaseTVAsync` method), 26  
[down\(\)](#) (`androidtv.basetv.basetv_sync.BaseTVSync` method), 34

## E

[enter\(\)](#) (`androidtv.basetv.basetv_async.BaseTVAsync` method), 26  
[enter\(\)](#) (`androidtv.basetv.basetv_sync.BaseTVSync` method), 34

## F

[FIRETV](#) (`androidtv.constants.DeviceEnum` attribute), 46  
[FireTVAsync](#) (class in `androidtv.firetv.firetv_async`), 41  
[FireTVSync](#) (class in `androidtv.firetv.firetv_sync`), 42  
[from\\_base\(\)](#) (`androidtv.androidtv.androidtv_async.AndroidTVAsync` class method), 12  
[from\\_base\(\)](#) (`androidtv.androidtv.androidtv_sync.AndroidTVSync` class method), 14  
[from\\_base\(\)](#) (`androidtv.firetv.firetv_async.FireTVAsync` class method), 41  
[from\\_base\(\)](#) (`androidtv.firetv.firetv_sync.FireTVSync` class method), 43

## G

[get\\_device\\_properties\(\)](#) (`androidtv.basetv.basetv_async.BaseTVAsync` method), 26  
[get\\_device\\_properties\(\)](#) (`androidtv.basetv.basetv_sync.BaseTVSync` method), 34

[get\\_hdmi\\_input\(\)](#) (`androidtv.basetv.basetv_async.BaseTVAsync` method), 26  
[get\\_hdmi\\_input\(\)](#) (`androidtv.basetv.basetv_sync.BaseTVSync` method), 35  
[get\\_installed\\_apps\(\)](#) (`androidtv.basetv.basetv_async.BaseTVAsync` method), 26  
[get\\_installed\\_apps\(\)](#) (`androidtv.basetv.basetv_sync.BaseTVSync` method), 35  
[get\\_properties\(\)](#) (`androidtv.androidtv.androidtv_async.AndroidTVAsync` method), 12  
[get\\_properties\(\)](#) (`androidtv.androidtv.androidtv_sync.AndroidTVSync` method), 14  
[get\\_properties\(\)](#) (`androidtv.firetv.firetv_async.FireTVAsync` method), 41  
[get\\_properties\(\)](#) (`androidtv.firetv.firetv_sync.FireTVSync` method), 43  
[get\\_properties\\_dict\(\)](#) (`androidtv.androidtv.androidtv_async.AndroidTVAsync` method), 13  
[get\\_properties\\_dict\(\)](#) (`androidtv.androidtv.androidtv_sync.AndroidTVSync` method), 14  
[get\\_properties\\_dict\(\)](#) (`androidtv.firetv.firetv_async.FireTVAsync` method), 42  
[get\\_properties\\_dict\(\)](#) (`androidtv.firetv.firetv_sync.FireTVSync` method), 44

## H

[HARDWARE\\_CUSTOMIZABLE\\_COMMANDS](#) (in module `androidtv.constants`), 46  
[ha\\_state\\_detection\\_rules\\_validator\(\)](#) (in module `androidtv`), 47  
[home\(\)](#) (`androidtv.basetv.basetv_async.BaseTVAsync` method), 26  
[home\(\)](#) (`androidtv.basetv.basetv_sync.BaseTVSync` method), 35

## I

[is\\_volume\\_muted\(\)](#) (`androidtv.basetv.basetv_async.BaseTVAsync` method), 26  
[is\\_volume\\_muted\(\)](#) (`androidtv.basetv.basetv_sync.BaseTVSync` method), 35

## K

method), 36

key\_r() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_r() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

key\_s() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_s() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

key\_t() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_t() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

key\_u() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_u() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

key\_v() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_v() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

key\_w() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_w() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

key\_x() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_x() (androidtv.basetv.basetv\_sync.BaseTVSync method), 36

key\_y() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_y() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

key\_z() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

key\_z() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

## L

launch\_app() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

launch\_app() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

learn\_sendevent() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

learn\_sendevent() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

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

left() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

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), 11

LockNotAcquiredException, 46

## M

media\_next\_track() (androidtv.basetv.basetv\_async.BaseTVAsync method), 28

media\_next\_track() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

media\_pause() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

media\_pause() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

media\_play() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

media\_play() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

media\_play\_pause() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

media\_play\_pause() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

media\_previous\_track() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

media\_previous\_track() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

media\_session\_state() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

media\_session\_state() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

MEDIA\_SESSION\_STATES (in module androidtv.constants), 46

media\_stop() (androidtv.basetv.basetv\_async.BaseTVAsync method), 29

media\_stop() (androidtv.basetv.basetv\_sync.BaseTVSync method), 37

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

[menu\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 37  
[mute\\_volume\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 29  
[mute\\_volume\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 37  
**P**  
[power\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 29  
[power\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 37  
[pull\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.AdbDeviceUsbAsync](#) method), 8  
[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.AdbDeviceUsbAsync](#) method), 8  
[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), 11  
**R**  
[right\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 29  
[right\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 38  
[running\\_apps\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 29  
[running\\_apps\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 38  
**S**  
[screen\\_on\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 29  
[screen\\_on\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 38  
[screen\\_on\\_awake\\_wake\\_lock\\_size\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 29  
[screen\\_on\\_awake\\_wake\\_lock\\_size\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 38  
[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  
[set\\_volume\\_level\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBPythonSync](#) method), 10  
[set\\_volume\\_level\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBServerSync](#) method), 11  
[set\\_volume\\_level\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 29  
[set\\_volume\\_level\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 38  
[setup\(\)](#) (in module [androidtv](#)), 48  
[setup\(\)](#) (in module [androidtv.setup\\_async](#)), 47  
[shell\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_async.AdbDeviceUsbAsync](#) method), 8  
[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), 10  
[shell\(\)](#) ([androidtv.adb\\_manager.adb\\_manager\\_sync.ADBServerSync](#) method), 11  
[sleep\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 30  
[sleep\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 38  
[space\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 30  
[space\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 38  
[start\\_intent\(\)](#) ([androidtv.basetv.basetv\\_async.BaseTVAsync](#) method), 30  
[start\\_intent\(\)](#) ([androidtv.basetv.basetv\\_sync.BaseTVSync](#) method), 38  
[state\\_detection\\_rules\\_validator\(\)](#) (in module [androidtv.basetv.basetv](#)), 22



`stop_app()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 30  
`stop_app()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 38  
`stream_music_properties()` (*an-*  
*droidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 30  
`stream_music_properties()` (*an-*  
*droidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 38  
`volume_level()` (*an-*  
*droidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 31  
`volume_level()` (*an-*  
*droidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 39  
`volume_up()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 31  
`volume_up()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 39

## T

`turn_off()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 30  
`turn_off()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 39  
`turn_on()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 30  
`turn_on()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 39  
`wake_lock_size()` (*an-*  
*droidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 31  
`wake_lock_size()` (*an-*  
*droidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 39

## W

## U

`up()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 30  
`up()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 39  
`update()` (*androidtv.androidtv\_async.AndroidTVAsync*  
*method*), 13  
`update()` (*androidtv.androidtv\_sync.AndroidTVSync*  
*method*), 15  
`update()` (*androidtv.firetv.firetv\_async.FireTVAsync*  
*method*), 42  
`update()` (*androidtv.firetv.firetv\_sync.FireTVSync*  
*method*), 44

## V

`VALID_PROPERTIES` (*in module androidtv.constants*),  
46  
`VALID_PROPERTIES_TYPES` (*in module an-*  
*droidtv.constants*), 46  
`VALID_STATE_PROPERTIES` (*in module an-*  
*droidtv.constants*), 46  
`VALID_STATES` (*in module androidtv.constants*), 46  
`volume()` (*androidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 30  
`volume()` (*androidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 39  
`volume_down()` (*an-*  
*droidtv.basetv.basetv\_async.BaseTVAsync*  
*method*), 30  
`volume_down()` (*an-*  
*droidtv.basetv.basetv\_sync.BaseTVSync*  
*method*), 39