

---

# **androidtv Documentation**

***Release 0.0.25***

**Jeff Irion**

**Aug 25, 2019**



# 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>19</b> |
| <b>4</b> | <b>ADB Intents and Commands</b>        | <b>21</b> |
| <b>5</b> | <b>Acknowledgments</b>                 | <b>23</b> |
| <b>6</b> | <b>Indices and tables</b>              | <b>25</b> |
|          | <b>Python Module Index</b>             | <b>27</b> |
|          | <b>Index</b>                           | <b>29</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: [python-adb](#)).

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.



## ANDROIDTV

## 2.1 androidtv package

### 2.1.1 Submodules

#### androidtv.androidtv module

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

ADB Debugging must be enabled.

```
class androidtv.androidtv.AndroidTV(host, adbkey="", adb_server_ip="",  
adb_server_port=5037, state_detection_rules=None)
```

Bases: *androidtv.basetv.BaseTV*

Representation of an Android TV device.

#### Parameters

- **host** (*str*) – The address of the device in the format <ip address>:<host>
- **adbkey** (*str*) – The path to the adbkey file for ADB authentication
- **adb\_server\_ip** (*str*) – The IP address of the ADB server
- **adb\_server\_port** (*int*) – The port for the ADB server
- **state\_detection\_rules** (*dict*, *None*) – A dictionary of rules for determining the state (see *BaseTV*)

**DEVICE\_CLASS** = 'androidtv'

**get\_properties** (*lazy=False*)

Get the properties needed for Home Assistant updates.

**Parameters** **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
- **audio\_state** (*str, None*) – The audio state, as determined from “`dumpsys audio`”, or `None` if it was not determined
- **device** (*str, None*) – The current 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

**get\_properties\_dict** (*lazy=True*)

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

**Parameters** **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', 'device', 'is\_volume\_muted', and 'volume'

**Return type** dict

**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>`

**turn\_off** ()

Send `POWER` action if the device is not off.

**turn\_on** ()

Send `POWER` action if the device is off.

**update** ()

Get the info needed for a Home Assistant update.

**Returns**

- **state** (*str*) – The state of the device
- **current\_app** (*str*) – The current running app
- **device** (*str*) – The current playback device
- **is\_volume\_muted** (*bool*) – Whether or not the volume is muted
- **volume\_level** (*float*) – The volume level (between 0 and 1)

## androidtv.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** (*host, 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': ['standby'],
                        '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': [{'playing': {'media_session_state
→': 3, 'wake_lock_size': 3}},
                                                {'paused': {'media_session_state
→': 3, 'wake_lock_size': 1}},
                                                'standby']}

```

The keys are app IDs, and the values are lists of rules that are evaluated in order by the `_custom_state_detection()` method.

#### VALID\_STATES

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

#### Valid rules:

- 'standby', 'playing', 'paused', 'idle', 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 in the format <ip address>:<host>
- **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)

#### `_adb_shell_pure_python_adb (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*

#### `_adb_shell_python_adb (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

**static** `_audio_state` (*dumpsys\_audio\_response*)

Parse the `audio_state` property from the output of `adb shell dumpsys audio`.

**Parameters** `dumpsys_audio_response` (*str*, *None*) – The output of `adb shell dumpsys audio`

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

**Return type** str, None

**static** `_conditions_are_true` (*conditions*, *media\_session\_state=None*, *wake\_lock\_size=None*, *audio\_state=None*)

Check whether the conditions in `conditions` are true.

**Parameters**

- **conditions** (*dict*) – A dictionary of conditions to be checked (see the `state_detection_rules` parameter in *BaseTV*)
- **media\_session\_state** (*int*, *None*) – The `media_session_state` property
- **wake\_lock\_size** (*int*, *None*) – The `wake_lock_size` property
- **audio\_state** (*str*, *None*) – The `audio_state` property

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

**Return type** bool

**static** `_current_app` (*current\_app\_response*)

Get the current app from the output of the command `androidtv.constants.CMD_CURRENT_APP`.

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

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

**Return type** str, None

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

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

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

**Returns**

- **current\_app** (*str*, *None*) – The current app, or *None* if it could not be determined
- **media\_session\_state** (*int*, *None*) – The state from the output of the ADB shell command, or *None* if it could not be determined

**\_custom\_state\_detection** (*current\_app=None*, *media\_session\_state=None*, *wake\_lock\_size=None*, *audio\_state=None*)

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

**Parameters**

- **current\_app** (*str*, *None*) – The `current_app` property
- **media\_session\_state** (*int*, *None*) – The `media_session_state` property
- **wake\_lock\_size** (*int*, *None*) – The `wake_lock_size` property
- **audio\_state** (*str*, *None*) – The `audio_state` property

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

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

**static** `_device` (*stream\_music*)

Get the current 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 playback device, or `None` if it could not be determined

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

**static** `_get_stream_music` (*dumpsys\_audio\_response=None*)

Get the `STREAM_MUSIC` block from `adb shell dumpsys audio`.

**Parameters** `dumpsys_audio_response` (*str*, *None*) – The output of `adb shell dumpsys audio`

**Returns** The `STREAM_MUSIC` block from `adb shell dumpsys audio`, or `None` if it could not be determined

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

**static** `_is_volume_muted` (*stream\_music*)

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

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

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

**Return type** `bool`, `None`

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

Send a key event to device.

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

**static** `_media_session_state` (*media\_session\_state\_response*, *current\_app*)

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

**Parameters**

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

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

**Return type** `int`, `None`

**static** `_running_apps` (*running\_apps\_response*)

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

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

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

**Return type** `list`, `None`

**\_volume** (*stream\_music, 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`
- **device** (*str, None*) – The current 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 audio\_state**

Check if audio is playing, paused, or idle.

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

**Return type** `str, None`

**property available**

Check whether the ADB connection is intact.

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

**Return type** `bool`

**property 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.

**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

**property 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

**property device**

Get the current playback device.

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

**Return type** str, None

**down()**

Send down action.

**enter()**

Send enter action.

**get\_device\_properties()**

Return a dictionary of device properties.

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

**Return type** dict

**home()**

Send home action.

**property 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.

**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).

**property 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.

**property running\_apps**  
Return a list of running user applications.

**Returns** A list of the running apps

**Return type** `list`

**property screen\_on**

Check if the screen is on.

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

**Return type** bool

**set\_volume\_level** (*volume\_level*, *current\_volume\_level=None*)

Set the volume to the desired level.

---

**Note:** This method works by sending volume up/down commands with a 1 second pause in between. Without this pause, the device will do a quick power cycle. This is the most robust solution I've found so far.

---

**Parameters**

- **volume\_level** (*float*) – The new volume level (between 0 and 1)
- **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*

**sleep** ()

Send sleep action.

**space** ()

Send space keypress.

**up** ()

Send up action.

**property 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*

**property 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*

**property** `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.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`).

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.constants module

Constants used in the [BaseTV](#), [AndroidTV](#), and [FireTV](#) classes.

### Links

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

`androidtv.constants.CMD_AUDIO_STATE` = `"dumpsys audio | grep -v 'Buffer Queue' | grep -q pa"`  
Get the audio state

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

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

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

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

`androidtv.constants.CMD_RUNNING_APPS` = `'ps | grep u0_a'`  
Get the running apps

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

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

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

`androidtv.constants.VALID_CONDITIONS` = ('audio\_state', 'media\_session\_state', 'wake\_lock\_state')  
 Conditions that can be checked by the `_conditions_are_true()` method

`androidtv.constants.VALID_PROPERTIES` = ('audio\_state', 'media\_session\_state')  
 Properties that can be used to determine the current state

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

## androidtv.firetv module

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

ADB Debugging must be enabled.

**class** `androidtv.firetv.FireTV`(*host*, *adbkey*="", *adb\_server\_ip*="", *adb\_server\_port*=5037, *state\_detection\_rules*=None)

Bases: `androidtv.basetv.BaseTV`

Representation of an Amazon Fire TV device.

### Parameters

- **host** (*str*) – The address of the device in the format <ip address>:<host>
- **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'

**\_send\_intent** (*pkg*, *intent*, *count*=1)

Send an intent to the device.

### Parameters

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

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

**Return type** dict

**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

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

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

**Parameters**

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

**Returns** A dictionary with keys 'screen\_on', 'awake', 'wake\_lock\_size', 'current\_app', 'media\_session\_state', and 'running\_apps'

**Return type** dict

**launch\_app** (*app*)

Launch an app.

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

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

**Return type** dict

**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*

**turn\_off** ()

Send `SLEEP` action if the device is not off.

**turn\_on** ()

Send `POWER` and `HOME` actions if the device is off.

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

Get the info needed for a Home Assistant update.

**Parameters** **get\_running\_apps** (*bool*) – Whether or not to get the *running\_apps* property

**Returns**

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

## 2.1.2 Module contents

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

ADB Debugging must be enabled.

`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, adbkey="", adb_server_ip="", adb_server_port=5037, state_detection_rules=None, device_class='auto')`

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 in the format `<ip address>:<host>`
- **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'`

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

**Return type** [AndroidTV](#), [FireTV](#)

`androidtv` is a Python 3 package that provides state information and control of Android TV and Fire TV devices via ADB. This package is used by the [Android TV](#) integration in Home Assistant.

## INSTALLATION

Be sure you install into a Python 3.x environment.

```
pip install androidtv
```





## 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 [python-adb](#) and [pure-python-adb](#).



## INDICES AND TABLES

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



## PYTHON MODULE INDEX

### a

- `androidtv`, [18](#)
- `androidtv.androidtv`, [5](#)
- `androidtv.basetv`, [6](#)
- `androidtv.constants`, [15](#)
- `androidtv.firetv`, [16](#)





## Symbols

`_adb_shell_pure_python_adb()` (*androidtv.basetv.BaseTV method*), 7  
`_adb_shell_python_adb()` (*androidtv.basetv.BaseTV method*), 7  
`_audio_state()` (*androidtv.basetv.BaseTV static method*), 8  
`_conditions_are_true()` (*androidtv.basetv.BaseTV static method*), 8  
`_current_app()` (*androidtv.basetv.BaseTV static method*), 8  
`_current_app_media_session_state()` (*androidtv.basetv.BaseTV method*), 8  
`_custom_state_detection()` (*androidtv.basetv.BaseTV method*), 8  
`_device()` (*androidtv.basetv.BaseTV static method*), 9  
`_get_stream_music()` (*androidtv.basetv.BaseTV method*), 9  
`_is_volume_muted()` (*androidtv.basetv.BaseTV static method*), 9  
`_key()` (*androidtv.basetv.BaseTV method*), 9  
`_media_session_state()` (*androidtv.basetv.BaseTV static method*), 9  
`_running_apps()` (*androidtv.basetv.BaseTV static method*), 9  
`_send_intent()` (*androidtv.firetv.FireTV method*), 16  
`_volume()` (*androidtv.basetv.BaseTV method*), 9  
`_volume_level()` (*androidtv.basetv.BaseTV method*), 10  
`_wake_lock_size()` (*androidtv.basetv.BaseTV static method*), 10

## A

AndroidTV (*class in androidtv.androidtv*), 5  
androidtv (*module*), 18  
androidtv.androidtv (*module*), 5  
androidtv.basetv (*module*), 6  
androidtv.constants (*module*), 15  
androidtv.firetv (*module*), 16  
audio\_state() (*androidtv.basetv.BaseTV property*), 10  
available() (*androidtv.basetv.BaseTV property*), 10

awake() (*androidtv.basetv.BaseTV property*), 10

## B

back() (*androidtv.basetv.BaseTV method*), 10  
BaseTV (*class in androidtv.basetv*), 6

## C

CMD\_AUDIO\_STATE (*in module androidtv.constants*), 15  
CMD\_AWAKE (*in module androidtv.constants*), 15  
CMD\_CURRENT\_APP (*in module androidtv.constants*), 15  
CMD\_MEDIA\_SESSION\_STATE (*in module androidtv.constants*), 15  
CMD\_MEDIA\_SESSION\_STATE\_FULL (*in module androidtv.constants*), 15  
CMD\_RUNNING\_APPS (*in module androidtv.constants*), 15  
CMD\_SCREEN\_ON (*in module androidtv.constants*), 15  
CMD\_WAKE\_LOCK\_SIZE (*in module androidtv.constants*), 15  
connect() (*androidtv.basetv.BaseTV method*), 10  
current\_app() (*androidtv.basetv.BaseTV property*), 11

## D

device() (*androidtv.basetv.BaseTV property*), 11  
DEVICE\_CLASS (*androidtv.androidtv.AndroidTV attribute*), 5  
DEVICE\_CLASS (*androidtv.firetv.FireTV attribute*), 16  
down() (*androidtv.basetv.BaseTV method*), 11

## E

enter() (*androidtv.basetv.BaseTV method*), 11

## F

FireTV (*class in androidtv.firetv*), 16

## G

get\_device\_properties() (*androidtv.basetv.BaseTV method*), 11

`get_properties()` (*androidtv.androidtv.AndroidTV method*), 5  
`get_properties()` (*androidtv.firetv.FireTV method*), 16  
`get_properties_dict()` (*androidtv.androidtv.AndroidTV method*), 6  
`get_properties_dict()` (*androidtv.firetv.FireTV method*), 17

## H

`ha_state_detection_rules_validator()` (*in module androidtv*), 18  
`home()` (*androidtv.basetv.BaseTV method*), 11

## I

`is_volume_muted()` (*androidtv.basetv.BaseTV property*), 11

## K

`key_0()` (*androidtv.basetv.BaseTV method*), 11  
`key_1()` (*androidtv.basetv.BaseTV method*), 11  
`key_2()` (*androidtv.basetv.BaseTV method*), 11  
`key_3()` (*androidtv.basetv.BaseTV method*), 11  
`key_4()` (*androidtv.basetv.BaseTV method*), 11  
`key_5()` (*androidtv.basetv.BaseTV method*), 11  
`key_6()` (*androidtv.basetv.BaseTV method*), 11  
`key_7()` (*androidtv.basetv.BaseTV method*), 11  
`key_8()` (*androidtv.basetv.BaseTV method*), 11  
`key_9()` (*androidtv.basetv.BaseTV method*), 11  
`key_a()` (*androidtv.basetv.BaseTV method*), 12  
`key_b()` (*androidtv.basetv.BaseTV method*), 12  
`key_c()` (*androidtv.basetv.BaseTV method*), 12  
`key_d()` (*androidtv.basetv.BaseTV method*), 12  
`key_e()` (*androidtv.basetv.BaseTV method*), 12  
`key_f()` (*androidtv.basetv.BaseTV method*), 12  
`key_g()` (*androidtv.basetv.BaseTV method*), 12  
`key_h()` (*androidtv.basetv.BaseTV method*), 12  
`key_i()` (*androidtv.basetv.BaseTV method*), 12  
`key_j()` (*androidtv.basetv.BaseTV method*), 12  
`key_k()` (*androidtv.basetv.BaseTV method*), 12  
`key_l()` (*androidtv.basetv.BaseTV method*), 12  
`key_m()` (*androidtv.basetv.BaseTV method*), 12  
`key_n()` (*androidtv.basetv.BaseTV method*), 12  
`key_o()` (*androidtv.basetv.BaseTV method*), 12  
`key_p()` (*androidtv.basetv.BaseTV method*), 12  
`key_q()` (*androidtv.basetv.BaseTV method*), 12  
`key_r()` (*androidtv.basetv.BaseTV method*), 12  
`key_s()` (*androidtv.basetv.BaseTV method*), 12  
`key_t()` (*androidtv.basetv.BaseTV method*), 12  
`key_u()` (*androidtv.basetv.BaseTV method*), 12  
`key_v()` (*androidtv.basetv.BaseTV method*), 13  
`key_w()` (*androidtv.basetv.BaseTV method*), 13  
`key_x()` (*androidtv.basetv.BaseTV method*), 13  
`key_y()` (*androidtv.basetv.BaseTV method*), 13

`key_z()` (*androidtv.basetv.BaseTV method*), 13

## L

`launch_app()` (*androidtv.firetv.FireTV method*), 17  
`left()` (*androidtv.basetv.BaseTV method*), 13

## M

`media_next_track()` (*androidtv.basetv.BaseTV method*), 13  
`media_pause()` (*androidtv.basetv.BaseTV method*), 13  
`media_play()` (*androidtv.basetv.BaseTV method*), 13  
`media_play_pause()` (*androidtv.basetv.BaseTV method*), 13  
`media_previous_track()` (*androidtv.basetv.BaseTV method*), 13  
`media_session_state()` (*androidtv.basetv.BaseTV property*), 13  
`MEDIA_SESSION_STATES` (*in module androidtv.constants*), 15  
`media_stop()` (*androidtv.basetv.BaseTV method*), 13  
`menu()` (*androidtv.basetv.BaseTV method*), 13  
`mute_volume()` (*androidtv.basetv.BaseTV method*), 13

## P

`power()` (*androidtv.basetv.BaseTV method*), 13

## R

`right()` (*androidtv.basetv.BaseTV method*), 13  
`running_apps()` (*androidtv.basetv.BaseTV property*), 13

## S

`screen_on()` (*androidtv.basetv.BaseTV property*), 13  
`set_volume_level()` (*androidtv.basetv.BaseTV method*), 14  
`setup()` (*in module androidtv*), 18  
`sleep()` (*androidtv.basetv.BaseTV method*), 14  
`space()` (*androidtv.basetv.BaseTV method*), 14  
`start_intent()` (*androidtv.androidtv.AndroidTV method*), 6  
`state_detection_rules_validator()` (*in module androidtv.basetv*), 15  
`stop_app()` (*androidtv.firetv.FireTV method*), 17

## T

`turn_off()` (*androidtv.androidtv.AndroidTV method*), 6  
`turn_off()` (*androidtv.firetv.FireTV method*), 17  
`turn_on()` (*androidtv.androidtv.AndroidTV method*), 6  
`turn_on()` (*androidtv.firetv.FireTV method*), 17

## U

`up()` (*androidtv.basetv.BaseTV method*), [14](#)

`update()` (*androidtv.androidtv.AndroidTV method*), [6](#)

`update()` (*androidtv.firetv.FireTV method*), [17](#)

## V

`VALID_CONDITIONS` (*in module androidtv.constants*), [16](#)

`VALID_PROPERTIES` (*in module androidtv.constants*), [16](#)

`VALID_STATES` (*in module androidtv.constants*), [16](#)

`volume()` (*androidtv.basetv.BaseTV property*), [14](#)

`volume_down()` (*androidtv.basetv.BaseTV method*), [14](#)

`volume_level()` (*androidtv.basetv.BaseTV property*), [14](#)

`volume_up()` (*androidtv.basetv.BaseTV method*), [14](#)

## W

`wake_lock_size()` (*androidtv.basetv.BaseTV property*), [15](#)