# androidtv Documentation

***Release 0.0.73***

**Jeff Irion**

**Apr 08, 2024**

# CONTENTS

CHAPTER

ONE

# 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
```

(continues on next page)

(continued from previous page)

```
  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
```

(continues on next page)

(continued from previous page)

```
  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.

CHAPTER

TWO

# ANDROIDTV

## 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(*log_errors=True*, *auth_timeout_s=10.0*, *transport_timeout_s=1.0*)

Connect to an Android TV / Fire TV device.

**Parameters**

- **log_errors** (*bool*) – Whether errors should be logged
- **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 static load_adbkey(*adbkey*)

Load the ADB keys.

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

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

**Return type**
PythonRSASigner

async pull(*local_path*, *device_path*)

Pull a file from the device using the Python ADB implementation.

**Parameters**

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

async push(*local_path*, *device_path*)

Push a file to the device using the Python ADB implementation.

**Parameters**

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

async screencap()

Take a screenshot using the Python ADB implementation.

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

**Return type**
bytes

async shell(*cmd*)

Send an ADB command using the Python ADB implementation.

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

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

**Return type**
str, None

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

Bases: `object`

A manager for ADB connections that uses an ADB server.

**Parameters**

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

**property available**

Check whether the ADB connection is intact.

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

**Return type**
bool

**async close**()

Close the ADB server socket connection.

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

**async connect**(*log_errors=True*)

Connect to an Android TV / Fire TV device.

**Parameters**
**log_errors** (*bool*) – Whether errors should be logged

**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*, *default_transport_timeout_s=None*, *banner=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.

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

class androidtv.adb_manager.adb_manager_sync.**ADBPythonSync**(*host*, *port*, *adbkey=''*, *signer=None*)

Bases: object

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

**Parameters**

- **host** (*str*) – The address of the device; may be an IP address or a host name
- **port** (*int*) – The device port to which we are connecting (default is 5555)
- **adbkey** (*str*) – The path to the adbkey file for ADB authentication
- **signer** (*PythonRSASigner, None*) – The signer for the ADB keys, as loaded by *ADBPythonSync.load_adbkey()*

**property available**

Check whether the ADB connection is intact.

**Returns**

Whether or not the ADB connection is intact

**Return type**
bool

`close()`

Close the ADB socket connection.

`connect(log_errors=True, auth_timeout_s=10.0, transport_timeout_s=1.0)`

Connect to an Android TV / Fire TV device.

**Parameters**

- **log_errors** (*bool*) – Whether errors should be logged
- **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

`screencap()`

Take a screenshot using the Python ADB implementation.

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

**Return type**
bytes

**shell**(*cmd*)

Send an ADB command using the Python ADB implementation.

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

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

**Return type**
str, None

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

Bases: object

A manager for ADB connections that uses an ADB server.

**Parameters**

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

**property available**

Check whether the ADB connection is intact.

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

**Return type**
bool

**close**()

Close the ADB server socket connection.

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

**connect**(*log_errors=True*)

Connect to an Android TV / Fire TV device.

**Parameters**
**log_errors** (*bool*) – Whether errors should be logged

**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

**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

**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: *BaseTVAsync*, *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: *BaseTVSync*, *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: *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 ADB shell output.

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

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

**Return type**
str, None

**_cmd_audio_state**()

Get the command used to retrieve the current audio state for this device.

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

**Return type**
str

`_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_hdmi_input()`

Get the command used to retrieve the current HDMI input for this device.

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

**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

**_cmd_volume_set**(*new_volume*)

Get the command used to set volume for this device.

**Parameters**
**new_volume** (*int*) – The new volume level

**Returns**
The device-specific ADB shell command used to set volume

**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 from the ADB shell output`.

**Parameters**
**hdmi_response** (*str, None*) – The output from the ADB command *androidtv.basetv.basetv.BaseTV._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
- **attribute** (*This method fills in the device_properties*) –
- **keys** (*which is a dictionary with*) –
- **'serialno'** –
- **'manufacturer'** –
- **'model'** –
- **'sw_version'** (*and*) –

`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: *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(*log_errors=True*, *auth_timeout_s=10.0*, *transport_timeout_s=1.0*)

Connect to an Android TV / Fire TV device.

**Parameters**

- **log_errors** (*bool*) – Whether errors should be logged
- **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_screencap()

Take a screencap.

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

**Returns**
The screencap 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, 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*, *adbkey=''*, *adb_server_ip=''*, *adb_server_port=5037*, *state_detection_rules=None*, *signer=None*)

Bases: *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**(*log_errors=True*, *auth_timeout_s=10.0*, *transport_timeout_s=1.0*)

Connect to an Android TV / Fire TV device.

**Parameters**

- **log_errors** (*bool*) – Whether errors should be logged
- **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_screencap**()

Take a screencap.

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

**Returns**
The screencap 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, 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: *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: *BaseTVAsync*, *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 `dumpsys media_session`, or `None` if it was not determined
- **running_apps** (*list, None*) – A list of the running apps, or `None` if it was not determined
- **hdmi_input** (*str, None*) – The HDMI input, or `None` if it could not be determined

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

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

**Parameters**

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

**Returns**

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

**Return type**

dict

`async 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*, *adbkey=''*, *adb_server_ip=''*, *adb_server_port=5037*, *state_detection_rules=None*, *signer=None*)

Bases: *BaseTVSync*, *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' && echo -e '1\\c' || (dumpsys audio | grep started | grep -qv 'Buffer Queue' && echo '2\\c' || echo '0\\c')"

Get the audio state

androidtv.constants.CMD_AUDIO_STATE11 = "CURRENT_AUDIO_STATE=$(dumpsys audio | sed -r -n '/[0-9]{2}-[0-9]{2}.*player piid:.*state:.*$/h; ${x;p;}') && echo $CURRENT_AUDIO_STATE | grep -q paused && echo -e '1\\c' || { echo $CURRENT_AUDIO_STATE | grep -q started && echo '2\\c' || echo '0\\c' ; }"

Get the audio state for an Android 11 device

androidtv.constants.CMD_AWAKE = 'dumpsys power | grep mWakefulness | grep -q Awake'

Determine whether the device is awake

androidtv.constants.CMD_CURRENT_APP = "CURRENT_APP=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp') && CURRENT_APP=${CURRENT_APP#*ActivityRecord{* * } && CURRENT_APP=${CURRENT_APP#*{* * } && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP%\\}*} && echo $CURRENT_APP"

Output identifier for current/focused application

androidtv.constants.CMD_CURRENT_APP11 = "CURRENT_APP=$(dumpsys window windows | grep -E -m 1 'mInputMethod(Input)?Target') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* } && echo $CURRENT_APP"

Output identifier for current/focused application for an Android 11 device

androidtv.constants.CMD_CURRENT_APP12 = "CURRENT_APP=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp|mObscuringWindow') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* } && echo $CURRENT_APP"

Output identifier for current/focused application for an Android 12 device

androidtv.constants.CMD_CURRENT_APP13 = "CURRENT_APP=$(dumpsys window windows | grep -E -m 1 'imeLayeringTarget|imeInputTarget|imeControlTarget') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* } && echo $CURRENT_APP"

Output identifier for current/focused application for an Android 13 device

androidtv.constants.CMD_CURRENT_APP_GOOGLE_TV = 'CURRENT_APP=$(dumpsys activity a . | grep mResumedActivity) && CURRENT_APP=${CURRENT_APP#*ActivityRecord{* * } && CURRENT_APP=${CURRENT_APP#*{* * } && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP%\\}*} && echo $CURRENT_APP'

Output identifier for current/focused application (for a Google TV device)

androidtv.constants.CMD_CURRENT_APP_MEDIA_SESSION_STATE = "CURRENT_APP=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp') && CURRENT_APP=${CURRENT_APP#*ActivityRecord{* * } && CURRENT_APP=${CURRENT_APP#*{* * } && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP%\\}*} && echo $CURRENT_APP && dumpsys media_session | grep -A 100 'Sessions Stack' | grep -A 100 $CURRENT_APP | grep -m 1 'state=PlaybackState {'"

Determine the current app and get the state from `dumpsys media_session`

androidtv.constants.CMD_CURRENT_APP_MEDIA_SESSION_STATE11 = "CURRENT_APP=$(dumpsys window windows | grep -E -m 1 'mInputMethod(Input)?Target') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* } && echo $CURRENT_APP && dumpsys media_session | grep -A 100 'Sessions Stack' | grep -A 100 $CURRENT_APP | grep -m 1 'state=PlaybackState {'"

Determine the current app and get the state from `dumpsys media_session` for an Android 11 device

androidtv.constants.CMD_CURRENT_APP_MEDIA_SESSION_STATE12 = "CURRENT_APP=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp|mObscuringWindow') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* } && echo $CURRENT_APP && dumpsys media_session | grep -A 100 'Sessions Stack' | grep -A 100 $CURRENT_APP | grep -m 1 'state=PlaybackState {'"

Determine the current app and get the state from `dumpsys media_session` for an Android 12 device

androidtv.constants.CMD_CURRENT_APP_MEDIA_SESSION_STATE13 = "CURRENT_APP=$(dumpsys window windows | grep -E -m 1 'imeLayeringTarget|imeInputTarget|imeControlTarget') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* } && echo $CURRENT_APP && dumpsys media_session | grep -A 100 'Sessions Stack' | grep -A 100 $CURRENT_APP | grep -m 1 'state=PlaybackState {'"

Determine the current app and get the state from `dumpsys media_session` for an Android 13 device

androidtv.constants.CMD_CURRENT_APP_MEDIA_SESSION_STATE_GOOGLE_TV = "CURRENT_APP=$(dumpsys activity a . | grep mResumedActivity) && CURRENT_APP=${CURRENT_APP#*ActivityRecord{* * } && CURRENT_APP=${CURRENT_APP#*{* * } && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP%\\}*} && echo $CURRENT_APP && dumpsys media_session | grep -A 100 'Sessions Stack' | grep -A 100 $CURRENT_APP | grep -m 1 'state=PlaybackState {'"

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=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp') && CURRENT_APP=${CURRENT_APP#*ActivityRecord{* * } && CURRENT_APP=${CURRENT_APP#*{* * } && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP%\\}*}"

Assign focused application identifier to `CURRENT_APP` variable

androidtv.constants.CMD_DEFINE_CURRENT_APP_VARIABLE11 = "CURRENT_APP=$(dumpsys window windows | grep -E -m 1 'mInputMethod(Input)?Target') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* }"

Assign focused application identifier to `CURRENT_APP` variable for an Android 11 device

androidtv.constants.CMD_DEFINE_CURRENT_APP_VARIABLE12 = "CURRENT_APP=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp|mObscuringWindow') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* }"

Assign focused application identifier to `CURRENT_APP` variable for an Android 12 device

androidtv.constants.CMD_DEFINE_CURRENT_APP_VARIABLE13 = "CURRENT_APP=$(dumpsys window windows | grep -E -m 1 'imeLayeringTarget|imeInputTarget|imeControlTarget') && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* }"

Assign focused application identifier to `CURRENT_APP` variable for an Android 13 device

androidtv.constants.CMD_DEFINE_CURRENT_APP_VARIABLE_GOOGLE_TV = 'CURRENT_APP=$(dumpsys activity a . | grep mResumedActivity) && CURRENT_APP=${CURRENT_APP#*ActivityRecord{* * } && CURRENT_APP=${CURRENT_APP#*{* * } && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP%\\}*}'

Assign focused application identifier to `CURRENT_APP` variable (for a Google TV device)

androidtv.constants.CMD_DEVICE_PROPERTIES = 'getprop ro.product.manufacturer && getprop ro.product.model && getprop ro.serialno && getprop ro.build.version.release'

The command used for getting the device properties

androidtv.constants.CMD_HDMI_INPUT = "dumpsys activity starter | grep -E -o '(ExternalTv|HDMI)InputService/HW[0-9]' -m 1 | grep -o 'HW[0-9]'"

Get the HDMI input

androidtv.constants.CMD_HDMI_INPUT11 = "(HDMI=$(dumpsys tv_input | grep 'ResourceClientProfile {.*}' | grep -o -E '(hdmi_port=[0-9]|TV)') && { echo ${HDMI/hdmi_port=/HW} | cut -d' ' -f1 ; }) || dumpsys activity starter | grep -E -o '(ExternalTv|HDMI)InputService/HW[0-9]' -m 1 | grep -o 'HW[0-9]'"

Get the HDMI input for an Android 11 device

androidtv.constants.CMD_INSTALLED_APPS = 'pm list packages'

Get installed apps

androidtv.constants.CMD_LAUNCH_APP = "CURRENT_APP=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp') && CURRENT_APP=${{CURRENT_APP#*ActivityRecord{{* * }} && CURRENT_APP=${{CURRENT_APP#*{{* * }} && CURRENT_APP=${{CURRENT_APP%%/*}} && CURRENT_APP=${{CURRENT_APP%\\}}*}} && if [ $CURRENT_APP != '{0}' ]; then monkey -p {0} -c android.intent.category.LEANBACK_LAUNCHER --pct-syskeys 0 1; fi"

Launch an app if it is not already the current app

androidtv.constants.CMD_LAUNCH_APP11 = "CURRENT_APP=$(dumpsys window windows | grep -E -m 1 'mInputMethod(Input)?Target') && CURRENT_APP=${{CURRENT_APP%%/*}} && CURRENT_APP=${{CURRENT_APP##* }} && if [ $CURRENT_APP != '{0}' ]; then monkey -p {0} -c android.intent.category.LEANBACK_LAUNCHER --pct-syskeys 0 1; fi"

Launch an app if it is not already the current app on an Android 11 device

androidtv.constants.CMD_LAUNCH_APP12 = "CURRENT_APP=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp|mObscuringWindow') && CURRENT_APP=${{CURRENT_APP%%/*}} && CURRENT_APP=${{CURRENT_APP##* }} && if [ $CURRENT_APP != '{0}' ]; then monkey -p {0} -c android.intent.category.LEANBACK_LAUNCHER --pct-syskeys 0 1; fi"

Launch an app if it is not already the current app on an Android 12 device

androidtv.constants.CMD_LAUNCH_APP13 = "CURRENT_APP=$(dumpsys window windows | grep -E -m 1 'imeLayeringTarget|imeInputTarget|imeControlTarget') && CURRENT_APP=${{CURRENT_APP%%/*}} && CURRENT_APP=${{CURRENT_APP##* }} && if [ $CURRENT_APP != '{0}' ]; then monkey -p {0} -c android.intent.category.LEANBACK_LAUNCHER --pct-syskeys 0 1; fi"

Launch an app if it is not already the current app on an Android 11 device

androidtv.constants.CMD_LAUNCH_APP_CONDITION = "if [ $CURRENT_APP != '{0}' ]; then monkey -p {0} -c android.intent.category.LEANBACK_LAUNCHER --pct-syskeys 0 1; fi"

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_CONDITION_FIRETV = "if [ $CURRENT_APP != '{0}' ]; then monkey -p {0} -c android.intent.category.LAUNCHER --pct-syskeys 0 1; fi"

Launch an app if it is not already the current app (assumes the variable CURRENT_APP has already been set) on a Fire TV

androidtv.constants.CMD_LAUNCH_APP_FIRETV = "CURRENT_APP=$(dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp') && CURRENT_APP=${{CURRENT_APP#*ActivityRecord{{* * }} && CURRENT_APP=${{CURRENT_APP#*{{* * }} && CURRENT_APP=${{CURRENT_APP%%/*}} && CURRENT_APP=${{CURRENT_APP%\\}}*}} && if [ $CURRENT_APP != '{0}' ]; then monkey -p {0} -c android.intent.category.LAUNCHER --pct-syskeys 0 1; fi"

Launch an app on a Fire TV device

androidtv.constants.CMD_LAUNCH_APP_GOOGLE_TV = "CURRENT_APP=$(dumpsys activity a . | grep mResumedActivity) && CURRENT_APP=${{CURRENT_APP#*ActivityRecord{{* * }} && CURRENT_APP=${{CURRENT_APP#*{{* * }} && CURRENT_APP=${{CURRENT_APP%%/*}} && CURRENT_APP=${{CURRENT_APP%\\}}*}} && if [ $CURRENT_APP != '{0}' ]; then monkey -p {0} -c android.intent.category.LEANBACK_LAUNCHER --pct-syskeys 0 1; fi"

Launch an app on a Google TV device

androidtv.constants.**CMD_MEDIA_SESSION_STATE = "dumpsys media_session | grep -A 100 'Sessions Stack' | grep -A 100 $CURRENT_APP | grep -m 1 'state=PlaybackState {'"**

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=${CURRENT_APP#*ActivityRecord{* * } && CURRENT_APP=${CURRENT_APP#*{* * } && CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP%\\}*}'**

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_PARSE_CURRENT_APP11 = 'CURRENT_APP=${CURRENT_APP%%/*} && CURRENT_APP=${CURRENT_APP##* }'**

Parse current application for an Android 11 device

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=ON' || dumpsys power | grep -q 'mScreenOn=true' || dumpsys display | grep -q 'mScreenState=ON')"**

Determine if the device is on

androidtv.constants.**CMD_SCREEN_ON_AWAKE_WAKE_LOCK_SIZE = "(dumpsys power | grep 'Display Power' | grep -q 'state=ON' || dumpsys power | grep -q 'mScreenOn=true' || dumpsys display | grep -q 'mScreenState=ON') && echo -e '1\\c' || echo -e '0\\c' && dumpsys power | grep mWakefulness | grep -q Awake && echo -e '1\\c' || echo -e '0\\c' && dumpsys power | grep Locks | grep 'size='"**

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 -q 'state=ON' || dumpsys power | grep -q 'mScreenOn=true' || dumpsys display | grep -q 'mScreenState=ON') && input keyevent 26"**

*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 'state=ON' || dumpsys power | grep -q 'mScreenOn=true' || dumpsys display | grep -q 'mScreenState=ON') && input keyevent 223"**

*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 'state=ON' || dumpsys power | grep -q 'mScreenOn=true' || dumpsys display | grep -q 'mScreenState=ON') || input keyevent 26"**

*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 'state=ON' || dumpsys power | grep -q 'mScreenOn=true' || dumpsys display | grep -q 'mScreenState=ON') || (input keyevent 26 && input keyevent 3)"**

*KEY_POWER = 26* and *KEY_HOME = 3* are defined below)

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

androidtv.constants.**CMD_VOLUME_SET_COMMAND = 'media volume --show --stream 3 --set {}'**

set volume

androidtv.constants.**CMD_VOLUME_SET_COMMAND11 = 'cmd media_session volume --show --stream 3 --set {}'**

set volume for an Android 11 & 12 & 13 device

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**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `IntEnum`

An enum for the various device types.

**ANDROIDTV = 1**

**BASETV = 0**

**FIRETV = 2**

androidtv.constants.**HA_CUSTOMIZABLE_COMMANDS = ('audio_state', 'current_app_media_session_state', 'hdmi_input', 'launch_app', 'running_apps', 'turn_off', 'turn_on')**

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_size')**

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 'int'>, 'wake_lock_size': <class 'int'>}**

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*, *log_errors=True*)

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)
- **log_errors** (*bool*) – Whether connection errors should be logged

**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*, *log_errors=True*)

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)
- **log_errors** (*bool*) – Whether connection errors should be logged

**Returns**
The representation of the device

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

CHAPTER

THREE

# 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]
```

CHAPTER

FOUR

# ADB INTENTS AND COMMANDS

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

CHAPTER

FIVE

# ACKNOWLEDGMENTS

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

CHAPTER

SIX

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## a

# INDEX

## Symbols

## A

## B



## C

## D



## E



## F



## G

## H



## I



## K

## L



## M

## P



## R



## S

## T



## U



## V



## W