

---

# **airtest Documentation**

**Game-Netease**

**Feb 21, 2019**



<b>1</b>	<b>Airtest</b>	<b>1</b>
1.1	Airtest	1
1.1.1	Getting Started	1
1.1.1.1	Supported Platforms	2
1.1.2	Installation	2
1.1.2.1	System Requirements	2
1.1.2.2	Installing the python package	2
1.1.3	Documentation	2
1.1.4	Example	3
1.1.5	Basic Usage	3
1.1.5.1	Connect Device	3
1.1.5.2	Simulate Input	4
1.1.5.3	Make Assertion	5
1.1.6	Running <code>.air</code> from CLI	5
1.1.6.1	run automated case	5
1.1.6.2	generate html report	5
1.1.6.3	get case info	6
1.1.7	Import from other <code>.air</code>	6
1.2	<code>airtest.core.api</code> module	6
1.3	<code>airtest.core.android</code> package	12
1.3.1	Submodules	12
1.3.1.1	<code>airtest.core.android.adb</code> module	12
1.3.1.2	<code>airtest.core.android.android</code> module	20
1.3.1.3	<code>airtest.core.android.constant</code> module	24
1.3.1.4	<code>airtest.core.android.ime</code> module	25
1.3.1.5	<code>airtest.core.android.javacap</code> module	25
1.3.1.6	<code>airtest.core.android.minicap</code> module	26
1.3.1.7	<code>airtest.core.android.minitouch</code> module	27
1.3.1.8	<code>airtest.core.android.recorder</code> module	31
1.3.1.9	<code>airtest.core.android.rotation</code> module	31
1.3.1.10	<code>airtest.core.android.yosemite</code> module	32
1.4	<code>airtest.core.ios</code> package	32
1.4.1	Submodules	33
1.4.1.1	<code>airtest.core.ios.ios</code> module	33
1.4.1.2	<code>airtest.core.ios.minicap</code> module	33
1.5	<code>airtest.core.win</code> package	34

1.5.1	Submodules . . . . .	34
1.5.1.1	airtest.core.win.screen module . . . . .	34
1.5.1.2	airtest.core.win.win module . . . . .	34
1.6	airtest . . . . .	34
1.6.1	airtest package . . . . .	34
1.6.1.1	Subpackages . . . . .	34
<b>Python Module Index</b>		<b>43</b>

### UI Automation Framework for Games and Apps

Airtest is a cross-platform UI automation framework for games and apps.

If you are new to Airtest, [Airtest Project Homepage](#) is a good place to get started.

The following documentation will guide you through main ideas of Airtest, as well as providing a API reference documentation.

## 1.1 Airtest

### Cross-Platform UI Automation Framework for Games and Apps

#### 1.1.1 Getting Started

- **Cross-Platform:** Airtest automates games and apps on almost all platforms.
- **Write Once, Run Anywhere:** Airtest provides cross-platform APIs, including app installation, simulated input, assertion and so forth. Airtest uses image recognition technology to locate UI elements, so that you can automate games and apps without injecting any code.
- **Fully Scalable:** Airtest cases can be easily run on large device farms, using commandline or python API. HTML reports with detailed info and screen recording allow you to quickly locate failure points. NetEase builds [Airlab](<https://airlab.163.com/>) on top of Airtest Project.
- **AirtestIDE:** AirtestIDE is an out of the box GUI tool that helps to create and run cases in a user-friendly way. AirtestIDE supports a complete automation workflow: `create -> run -> report`.

[Get Started from Airtest Project Homepage](#)

### 1.1.1.1 Supported Platforms

- Android
- iOS
- Windows
- Unity
- Cocos2dx
- Egret
- WeChat

### 1.1.2 Installation

This section describes how to install Airtest python library. Download AirtestIDE from our [homepage](#) if you need to use the GUI tool.

#### 1.1.2.1 System Requirements

- Operating System:
  - Windows
  - MacOS X
  - Linux
- Python2.7 & Python3.3+

#### 1.1.2.2 Installing the python package

Airtest package can be installed directly from Pypi. Use `pip` to manage installation of all python dependencies and package itself.

```
pip install -U airtest
```

You can also install it from Git repository.

```
git clone https://github.com/AirtestProject/Airtest.git
pip install -e airtest
```

Use `-e` here to install airtest in develop mode since this repo is in rapid development. Then you can upgrade the repo with `git pull` later.

### 1.1.3 Documentation

You can find the complete Airtest documentation on [readthedocs](#).

### 1.1.4 Example

Airtest provides simple APIs that are platform independent. This section describes how to create an automated case which does the following:

1. connects to local android device with adb
2. installs the apk application
3. runs application and takes the screenshot
4. performs several user operations (touch, swipe, keyevent)
5. uninstalls application

```
from airtest.core.api import *

# connect an android phone with adb
init_device("Android")
# or use connect_device api
# connect_device("Android:///")

install("path/to/your/apk")
start_app("package_name_of_your_apk")
touch(Template("image_of_a_button.png"))
swipe(Template("slide_start.png"), Template("slide_end.png"))
assert_exists(Template("success.png"))
keyevent("BACK")
home()
uninstall("package_name_of_your_apk")
```

For more detailed info, please refer to [Airtest Python API reference](#) or take a look at [API code](#)

### 1.1.5 Basic Usage

Airtest aims at providing platform independent API, so that you can write automated cases once and run it on multiple devices and platforms.

1. Using `connect_device` API you can connect to any android/iOS device or windows application.
2. Then perform `simulated input` to automate your game or app.
3. **DO NOT** forget to `make assertions` of the expected result.

#### 1.1.5.1 Connect Device

Using `connect_device` API you can connect to any android/iOS device or windows application.

```
connect_device("platform://host:port/uuid?param=value&param2=value2")
```

- platform: Android/iOS/Windows...
- host: adb host for android, iproxy host for iOS, empty for other platforms
- port: adb port for android, iproxy port for iOS, empty for other platforms
- uuid: uuid for target device, e.g. serialno for Android, handle for Windows, uuid for iOS
- param: device initialization configuration fields. e.g. cap\_method/ori\_method/...
- value: device initialization configuration field values.

see also `connect_device`.

### Connect android device

1. Connect your android phone to your PC with usb
2. Use `adb devices` to make sure the state is device
3. Connect device in Airtest
4. If you have multiple devices or even remote devices, use more params to specify the device

```
# connect an android phone with adb
init_device("Android")

# or use connect_device api with default params
connect_device("android:///")

# connect a remote device using custom params
connect_device("android://adbhost:adbport/1234566?cap_method=javacap&touch_method=adb
↪")
```

### Connect iOS device

Follow the instruction of `iOS-Tagent` to setup the environment.

```
# connect a local ios device
connect_device("ios:///")
```

### Connect windows application

```
# connect local windows desktop
connect_device("Windows:///")

# connect local windows application
connect_device("Windows:///?title_re=unity.*")
```

Airtest uses `pywinauto` as Windows backend. For more window searching params, please see `pywinauto` documentation.

#### 1.1.5.2 Simulate Input

Following APIs are fully supported:

- touch
- swipe
- text
- keyevent
- snapshot
- wait



More APIs are available, some of which may be platform specific, please see [API reference](#) for more information.

### 1.1.5.3 Make Assertion

Airtest provide some assert functions, including:

- `assert_exists`
- `assert_not_exists`
- `assert_equal`
- `assert_not_equal`

When assertion fails, it will raise `AssertionError`. And you will see all assertions in the html report.

### 1.1.6 Running `.air` from CLI

Using AirtestIDE, you can easily create and author automated cases as `.air` directories. Airtest CLI provides the possibility to execute cases on different host machine and target device platforms without using AirtestIDE itself.

Connections to devices are specified by command line arguments, i.e. the code is platform independent and one automated case can be used for Android, iOS or Windows apps as well.

Following examples demonstrate the basic usage of airtest framework running from CLI. For a deeper understanding, try running provided automated cases: `airtest/playground/test_blackjack.air`

#### 1.1.6.1 run automated case

```
# run automated cases and scenarios on various devices
> airtest run "path to your .air dir" --device Android:///
> airtest run "path to your .air dir" --device Android://adbhost:adbport/serialno
> airtest run "path to your .air dir" --device Windows:///?title_re=Unity.*
> airtest run "path to your .air dir" --device iOS:///
...
# show help
> airtest run -h
usage: airtest run [-h] [--device [DEVICE]] [--log [LOG]]
                  [--recording [RECORDING]]
                  script
positional arguments:
  script                air path
optional arguments:
  -h, --help            show this help message and exit
  --device [DEVICE]    connect dev by uri string, e.g. Android:///
  --log [LOG]          set log dir, default to be script dir
  --recording [RECORDING]
                        record screen when running
```

#### 1.1.6.2 generate html report

```
> airtest report "path to your .air dir"
log.html
> airtest report -h
usage: airtest report [-h] [--outfile OUTFILE] [--static_root STATIC_ROOT]
                    [--log_root LOG_ROOT] [--record RECORD [RECORD ...]]
                    [--export EXPORT] [--lang LANG]
                    script

positional arguments:
  script                script filepath

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE    output html filepath, default to be log.html
  --static_root STATIC_ROOT
                        static files root dir
  --log_root LOG_ROOT  log & screen data root dir, logfile should be
                        log_root/log.txt
  --record RECORD [RECORD ...]
                        custom screen record file path
  --export EXPORT      export a portable report dir containing all resources
  --lang LANG          report language
```

### 1.1.6.3 get case info

```
# print case info in json if defined, including: author, title, desc
> python -m airtest info "path to your .air dir"
{"author": ..., "title": ..., "desc": ...}
```

### 1.1.7 Import from other .air

You can write some common used function in one .air script and import it from other scripts. Airtest provide using API to manage the context change including `sys.path` and Template search path.

```
from airtest.core.api import using
using("common.air")

from common import common_function

common_function()
```

## 1.2 airtest.core.api module

This module contains the Airtest Core APIs.

**init\_device** (*platform='Android', uuid=None, \*\*kwargs*)  
Initialize device if not yet, and set as current device.

#### Parameters

- **platform** – Android, IOS or Windows
- **uuid** – uuid for target device, e.g. serialno for Android, handle for Windows, uuid for iOS

- **kwargs** – Optional platform specific keyword args, e.g. `cap_method=JAVACAP` for Android

**Returns** device instance

**connect\_device** (*uri*)

Initialize device with uri, and set as current device.

**Parameters** **uri** – an URI where to connect to device, e.g. `android://adbhost:adbport/serialno?param=value&param2=value2`

**Returns** device instance

**Example**

- `android:///` # local adb device using default params
- `android://adbhost:adbport/1234566?cap_method=javacap&touch_method=adb`  
# remote device using custom params
- `windows:///` # local Windows application
- `ios:///` # iOS device

**device** ()

Return the current active device.

**Returns** current device instance

**set\_current** (*idx*)

Set current active device.

**Parameters** **idx** – uuid or index of initialized device instance

**Raises** **IndexError** – raised when device idx is not found

**Returns** None

**Platforms** Android, iOS, Windows

**auto\_setup** (*basedir=None, devices=None, logdir=None, project\_root=None*)

Auto setup running env and try connect android device if not device connected. :param basedir: basedir of script, `__file__` is also acceptable. :param devices: connect\_device uri in list. :param logdir: log dir for script report, default is None for no log, set to `True` for `<basedir>/log`. :param project\_root: project root dir for `using` api.

**shell** (*\*args, \*\*kwargs*)

Start remote shell in the target device and execute the command

**Parameters** **cmd** – command to be run on device, e.g. `"ls /data/local/tmp"`

**Returns** the output of the shell cmd

**Platforms** Android

**start\_app** (*\*args, \*\*kwargs*)

Start the target application on device

**Parameters**

- **package** – name of the package to be started, e.g. `"com.netease.my"`
- **activity** – the activity to start, default is None which means the main activity

**Returns** None

**Platforms** Android, iOS

**stop\_app** (\*args, \*\*kwargs)

Stop the target application on device

**Parameters** **package** – name of the package to stop, see also *start\_app*

**Returns** None

**Platforms** Android, iOS

**clear\_app** (\*args, \*\*kwargs)

Clear data of the target application on device

**Parameters** **package** – name of the package, see also *start\_app*

**Returns** None

**Platforms** Android, iOS

**install** (\*args, \*\*kwargs)

Install application on device

**Parameters** **filepath** – the path to file to be installed on target device

**Returns** None

**Platforms** Android, iOS

**uninstall** (\*args, \*\*kwargs)

Uninstall application on device

**Parameters** **package** – name of the package, see also *start\_app*

**Returns** None

**Platforms** Android, iOS

**snapshot** (\*args, \*\*kwargs)

Take the screenshot of the target device and save it to the file.

**Parameters**

- **filename** – name of the file where to save the screenshot. If the relative path is provided, the default location is `ST.LOG_DIR`
- **msg** – short description for screenshot, it will be recorded in the report

**Returns** absolute path of the screenshot

**Platforms** Android, iOS, Windows

**wake** (\*args, \*\*kwargs)

Wake up and unlock the target device

**Returns** None

**Platforms** Android, iOS

---

**Note:** Might not work on some models

---

**home** (\*args, \*\*kwargs)

Return to the home screen of the target device.

**Returns** None

**Platforms** Android, iOS

**touch** (\*args, \*\*kwargs)

Perform the touch action on the device screen

**Parameters**

- **v** – target to touch, either a Template instance or absolute coordinates (x, y)
- **times** – how many touches to be performed
- **kwargs** – platform specific *kwargs*, please refer to corresponding docs

**Returns** final position to be clicked

**Platforms** Android, Windows, iOS

**click** (\*args, \*\*kwargs)

Perform the touch action on the device screen

**Parameters**

- **v** – target to touch, either a Template instance or absolute coordinates (x, y)
- **times** – how many touches to be performed
- **kwargs** – platform specific *kwargs*, please refer to corresponding docs

**Returns** final position to be clicked

**Platforms** Android, Windows, iOS

**double\_click** (\*args, \*\*kwargs)

**swipe** (\*args, \*\*kwargs)

Perform the swipe action on the device screen.

**There are two ways of assigning the parameters**

- `swipe(v1, v2=Template(...))` # swipe from v1 to v2
- `swipe(v1, vector=(x, y))` # swipe starts at v1 and moves along the vector.

**Parameters**

- **v1** – the start point of swipe, either a Template instance or absolute coordinates (x, y)
- **v2** – the end point of swipe, either a Template instance or absolute coordinates (x, y)
- **vector** – a vector coordinates of swipe action, either absolute coordinates (x, y) or percentage of screen e.g.(0.5, 0.5)
- **\*\*kwargs** – platform specific *kwargs*, please refer to corresponding docs

**Raises Exception** – general exception when not enough parameters to perform swap action have been provided

**Returns** Origin position and target position

**Platforms** Android, Windows, iOS

**pinch** (\*args, \*\*kwargs)

Perform the pinch action on the device screen

**Parameters**

- **in\_or\_out** – pinch in or pinch out, enum in ["in", "out"]
- **center** – center of pinch action, default as None which is the center of the screen

- **percent** – percentage of the screen of pinch action, default is 0.5

**Returns** None

**Platforms** Android

**keyevent** (*\*args*, *\*\*kwargs*)

Perform key event on the device

**Parameters**

- **keyname** – platform specific key name
- **\*\*kwargs** – platform specific *kwargs*, please refer to corresponding docs

**Returns** None

**Platforms** Android, Windows, iOS

**text** (*\*args*, *\*\*kwargs*)

Input text on the target device. Text input widget must be active first.

**Parameters**

- **text** – text to input, unicode is supported
- **enter** – input *Enter* keyevent after text input, default is True

**Returns** None

**Platforms** Android, Windows, iOS

**sleep** (*\*args*, *\*\*kwargs*)

Set the sleep interval. It will be recorded in the report

**Parameters** **secs** – seconds to sleep

**Returns** None

**Platforms** Android, Windows, iOS

**wait** (*\*args*, *\*\*kwargs*)

Wait to match the Template on the device screen

**Parameters**

- **v** – target object to wait for, Template instance
- **timeout** – time interval to wait for the match, default is None which is `ST.FIND_TIMEOUT`
- **interval** – time interval in seconds to attempt to find a match
- **intervalfunc** – called after each unsuccessful attempt to find the corresponding match

**Raises** *TargetNotFoundError* – raised if target is not found after the time limit expired

**Returns** coordinates of the matched target

**Platforms** Android, Windows, iOS

**exists** (*\*args*, *\*\*kwargs*)

Check whether given target exists on device screen

**Parameters** **v** – target to be checked

**Returns** False if target is not found, otherwise returns the coordinates of the target

**Platforms** Android, Windows, iOS

**find\_all** (\*args, \*\*kwargs)

Find all occurrences of the target on the device screen and return their coordinates

**Parameters** **v** – target to find

**Returns** list of coordinates, [(x, y), (x1, y1), ...]

**Platforms** Android, Windows, iOS

**assert\_exists** (\*args, \*\*kwargs)

Assert target exists on device screen

**Parameters**

- **v** – target to be checked
- **msg** – short description of assertion, it will be recorded in the report

**Raises** **AssertionError** – if assertion fails

**Returns** coordinates of the target

**Platforms** Android, Windows, iOS

**assert\_not\_exists** (\*args, \*\*kwargs)

Assert target does not exist on device screen

**Parameters**

- **v** – target to be checked
- **msg** – short description of assertion, it will be recorded in the report

**Raises** **AssertionError** – if assertion fails

**Returns** None.

**Platforms** Android, Windows, iOS

**assert\_equal** (\*args, \*\*kwargs)

Assert two values are equal

**Parameters**

- **first** – first value
- **second** – second value
- **msg** – short description of assertion, it will be recorded in the report

**Raises** **AssertionError** – if assertion fails

**Returns** None

**Platforms** Android, Windows, iOS

**assert\_not\_equal** (\*args, \*\*kwargs)

Assert two values are not equal

**Parameters**

- **first** – first value
- **second** – second value
- **msg** – short description of assertion, it will be recorded in the report

**Raises** **AssertionError** – if assertion

**Returns** None

**Platforms** Android, Windows, iOS

## 1.3 airtest.core.android package

This package provide Android Device Class.

### 1.3.1 Submodules

#### 1.3.1.1 airtest.core.android.adb module

**class** **ADB** (*serialno=None, adb\_path=None, server\_addr=None*)

Bases: object

adb client object class

**status\_device** = 'device'

**status\_offline** = 'offline'

**SHELL\_ENCODING** = 'utf-8'

**static builtin\_adb\_path** ()

Return built-in adb executable path

**Returns** adb executable path

**start\_server** ()

Perform *adb start-server* command to start the adb server

**Returns** None

**kill\_server** ()

Perform *adb kill-server* command to kill the adb server

**Returns** None

**version** ()

Perform *adb version* command and return the command output

**Returns** command output

**start\_cmd** (*cmds, device=True*)

Start a subprocess with adb command(s)

**Parameters**

- **cmds** – command(s) to be run
- **device** – if True, the device serial number must be specified by *-s serialno* argument

**Raises** `RuntimeError` – if *device* is True and *serialno* is not specified

**Returns** a subprocess

**cmd** (*cmds, device=True, ensure\_unicode=True*)

Run the adb command(s) in subprocess and return the standard output

**Parameters**

- **cmds** – command(s) to be run
- **device** – if True, the device serial number must be specified by *-s serialno* argument



- **ensure\_unicode** – encode/decode unicode of standard outputs (stdout, stderr)

**Raises**

- `DeviceConnectionError` – if any error occurs when connecting the device
- `AdbError` – if any other adb error occurs

**Returns** command(s) standard output (stdout)

**close\_proc\_pipe** (*proc*)

close stdin/stdout/stderr of subprocess.Popen.

**devices** (*state=None*)

Perform *adb devices* command and return the list of adb devices

**Parameters** **state** – optional parameter to filter devices in specific state

**Returns** list of adb devices

**connect** (*force=False*)

Perform *adb connect* command, remote devices are preferred to connect first

**Parameters** **force** – force connection, default is False

**Returns** None

**disconnect** ()

Perform *adb disconnect* command

**Returns** None

**get\_status** ()

Perform *adb get-state* and return the device status

**Raises** `AdbError` – if status cannot be obtained from the device

**Returns** None if status is *not found*, otherwise return the standard output from *adb get-state* command

**wait\_for\_device** (*timeout=5*)

Perform *adb wait-for-device* command

**Parameters** **timeout** – time interval in seconds to wait for device

**Raises** `DeviceConnectionError` – if device is not available after timeout

**Returns** None

**start\_shell** (*cmds*)

Handle *adb shell* command(s)

**Parameters** **cmds** – adb shell command(s)

**Returns** None

**raw\_shell** (*cmds, ensure\_unicode=True*)

Handle *adb shell* command(s) with unicode support

**Parameters**

- **cmds** – adb shell command(s)
- **ensure\_unicode** – decode/encode unicode True or False, default is True

**Returns** command(s) output

**shell** (*cmd*)

Run the *adb shell* command on the device

**Parameters** **cmd** – a command to be run

**Raises** `AdbShellError` – if command return value is non-zero or if any other *AdbError* occurred

**Returns** command output

**keyevent** (*keyname*)

Perform *adb shell input keyevent* command on the device

**Parameters** **keyname** – key event name

**Returns** None

**getprop** (*key, strip=True*)

Perform *adb shell getprop* on the device

**Parameters**

- **key** – key value for property
- **strip** – True or False to strip the return carriage and line break from returned string

**Returns** property value

**sdk\_version**

Get the SDK version from the device

**Returns** SDK version

**push** (*local, remote*)

Perform *adb push* command

**Parameters**

- **local** – local file to be copied to the device
- **remote** – destination on the device where the file will be copied

**Returns** None

**pull** (*remote, local*)

Perform *adb pull* command :param remote: remote file to be downloaded from the device :param local: local destination where the file will be downloaded from the device

**Returns** None

**forward** (*local, remote, no\_rebind=True*)

Perform *adb forward* command

**Parameters**

- **local** – local tcp port to be forwarded
- **remote** – tcp port of the device where the local tcp port will be forwarded
- **no\_rebind** – True or False

**Returns** None

**get\_forwards** ()

Perform *'adb forward -list'* command

**Yields** serial number, local tcp port, remote tcp port

**Returns** None

**classmethod** `get_available_forward_local()`

Generate a pseudo random number between 11111 and 20000 that will be used as local forward port

**Returns** integer between 11111 and 20000

---

**Note:** use `forward -no-rebind` to check if port is available

---

**setup\_forward** (*\*\*kwargs*)

**remove\_forward** (*local=None*)

Perform `adb forward -remove` command

**Parameters** `local` – local tcp port

**Returns** None

**install\_app** (*filepath, replace=False*)

Perform `adb install` command

**Parameters**

- **filepath** – full path to file to be installed on the device
- **replace** – force to replace existing application, default is False

**Returns** command output

**install\_multiple\_app** (*filepath, replace=False*)

Perform `adb install-multiple` command

**Parameters**

- **filepath** – full path to file to be installed on the device
- **replace** – force to replace existing application, default is False

**Returns** command output

**pm\_install** (*filepath, replace=False*)

Perform `adb push` and `adb install` commands

---

**Note:** This is more reliable and recommended way of installing `.apk` files

---

**Parameters**

- **filepath** – full path to file to be installed on the device
- **replace** – force to replace existing application, default is False

**Returns** None

**uninstall\_app** (*package*)

Perform `adb uninstall` command :param package: package name to be uninstalled from the device

**Returns** command output

**pm\_uninstall** (*package, keepdata=False*)

Perform `adb uninstall` command and delete all related application data

**Parameters**

- **package** – package name to be uninstalled from the device
- **keepdata** – True or False, keep application data after removing the app from the device

**Returns** command output

**snapshot** ()

Take the screenshot of the device display

**Returns** command output (stdout)

**touch** (*tuple\_xy*)

Perform user input (touchscreen) on given coordinates

**Parameters** **tuple\_xy** – coordinates (x, y)

**Returns** None

**swipe** (*tuple\_x0y0*, *tuple\_x1y1*, *duration=500*)

Perform user input (swipe screen) from start point (x,y) to end point (x,y)

**Parameters**

- **tuple\_x0y0** – start point coordinates (x, y)
- **tuple\_x1y1** – end point coordinates (x, y)
- **duration** – time interval for action, default 500

**Raises** `AirtestError` – if SDK version is not supported

**Returns** None

**logcat** (*grep\_str=""*, *extra\_args=""*, *read\_timeout=10*)

Perform `adb shell logcat` command and search for given patterns

**Parameters**

- **grep\_str** – pattern to filter from the logcat output
- **extra\_args** – additional logcat arguments
- **read\_timeout** – time interval to read the logcat, default is 10

**Yields** logcat lines containing filtered patterns

**Returns** None

**exists\_file** (*filepath*)

Check if the file exists on the device

**Parameters** **filepath** – path to the file

**Returns** True or False if file found or not

**file\_size** (*filepath*)

Get the file size

**Parameters** **filepath** – path to the file

**Returns** The file size

**line\_breaker**

Set carriage return and line break property for various platforms and SDK versions

**Returns** carriage return and line break string

**display\_info**

Set device display properties (orientation, rotation and max values for x and y coordinates)

Notes: if there is a lock screen detected, the function tries to unlock the device first

**Returns** device screen properties

**get\_display\_info()**

Get information about device physical display (orientation, rotation and max values for x and y coordinates)

**Returns** device screen properties

**getMaxXY()**

Get device display maximum values for x and y coordinates

**Returns** max x and max y coordinates

**getRestrictedScreen()**

Get value for mRestrictedScreen (without black border / virtual keyboard)

**Returns** screen resolution mRestrictedScreen value as tuple (x, y)

**getPhysicalDisplayInfo()**

Get value for display dimension and density from *mPhysicalDisplayInfo* value obtained from *dumpsys* command.

**Returns** physical display info for dimension and density

**getDisplayOrientation()**

Another way to get the display orientation, this works well for older devices (SDK version 15)

**Returns** display orientation information

**get\_top\_activity()**

Perform *adb shell dumpsys activity top* command search for the top activity

**Raises** `AirtestError` – if top activity cannot be obtained

**Returns** top activity as a tuple

**is\_keyboard\_shown()**

Perform *adb shell dumpsys input\_method* command and search for information if keyboard is shown

**Returns** True or False whether the keyboard is shown or not

**is\_screenon()**

Perform *adb shell dumpsys window policy* command and search for information if screen is turned on or off

**Raises** `AirtestError` – if screen state can't be detected

**Returns** True or False whether the screen is turned on or off

**is\_locked()**

Perform *adb shell dumpsys window policy* command and search for information if screen is locked or not

**Raises** `AirtestError` – if lock screen can't be detected

**Returns** True or False whether the screen is locked or not

**Notes**

Does not work on Xiaomi 2S

**unlock()**

Perform *adb shell input keyevent MENU* and *adb shell input keyevent BACK* commands to attempt to unlock the screen

**Returns** None

**Warning:** Might not work on all devices

**get\_package\_version(package)**

Perform *adb shell dumpsys package* and search for information about given package version

**Parameters** **package** – package name

**Returns** None if no info has been found, otherwise package version

**list\_app(third\_only=False)**

**Perform** *adb shell pm list packages* to print all packages, optionally only those whose package name contains the text in **FILTER**.

**Options** -f: see their associated file -d: filter to only show disabled packages -e: filter to only show enabled packages -s: filter to only show system packages -3: filter to only show third party packages -i: see the installer for the packages -u: also include uninstalled packages

**Parameters** **third\_only** – print only third party packages

**Returns** list of packages

**path\_app(package)**

Perform *adb shell pm path* command to print the path to the package

**Parameters** **package** – package name

**Raises**

- `AdbShellError` – if any adb error occurs
- `AirtestError` – if package is not found on the device

**Returns** path to the package

**check\_app(package)**

Perform *adb shell dumpsys package* command and check if package exists on the device

**Parameters** **package** – package name

**Raises** `AirtestError` – if package is not found

**Returns** True if package has been found

**start\_app(package, activity=None)**

Perform *adb shell monkey* commands to start the application, if *activity* argument is *None*, then *adb shell am start* command is used.

**Parameters**

- **package** – package name
- **activity** – activity name

**Returns** None

**start\_app\_timing(package, activity)**

Start the application and activity, and measure time

**Parameters**

- **package** – package name
- **activity** – activity name

**Returns** app launch time

**stop\_app** (*package*)

Perform *adb shell am force-stop* command to force stop the application

**Parameters** **package** – package name

**Returns** None

**clear\_app** (*package*)

Perform *adb shell pm clear* command to clear all application data

**Parameters** **package** – package name

**Returns** None

**get\_ip\_address** ()

**Perform several set of commands to obtain the IP address**

- *adb shell netcfg | grep wlan0*
- *adb shell ifconfig*
- *adb getprop dhcp.wlan0.ipaddress*

**Returns** None if no IP address has been found, otherwise return the IP address

**get\_gateway\_address** ()

**Perform several set of commands to obtain the gateway address**

- *adb getprop dhcp.wlan0.gateway*
- *adb shell netcfg | grep wlan0*

**Returns** None if no gateway address has been found, otherwise return the gateway address

**get\_memory** ()**get\_storage** ()**get\_cpuinfo** ()**get\_cpufreq** ()**get\_gpu** ()**get\_model** ()**get\_manufacturer** ()**get\_device\_info** ()

Get android device information, including: memory/storage/display/cpu/gpu/model/manufacturer...

**Returns** Dict of info

**cleanup\_adb\_forward** ()

### 1.3.1.2 airtest.core.android.android module

```
class Android (serialno=None, host=None, cap_method='MINICAP_STREAM',  
                touch_method='MINITOUCH', ime_method='YOSEMITEIME',  
                ori_method='MINICAPORI')
```

Bases: *airtest.core.device.Device*

Android Device Class

**get\_default\_device** ()

Get local default device when no serialno

**Returns** local device serialno

**uuid**

**list\_app** (*third\_only=False*)

Return list of packages

**Parameters** **third\_only** – if True, only third party applications are listed

**Returns** array of applications

**path\_app** (*package*)

Print the full path to the package

**Parameters** **package** – package name

**Returns** the full path to the package

**check\_app** (*package*)

Check is package exists on the device

**Parameters** **package** – package name

**Returns** True or False whether the package exists on the device or not

**start\_app** (*package, activity=None*)

Start the application and activity

**Parameters**

- **package** – package name
- **activity** – activity name

**Returns** None

**start\_app\_timing** (*package, activity*)

Start the application and activity, and measure time

**Parameters**

- **package** – package name
- **activity** – activity name

**Returns** app launch time

**stop\_app** (*package*)

Stop the application

**Parameters** **package** – package name

**Returns** None

**clear\_app** (*package*)

Clear all application data



**Parameters** `package` – package name

**Returns** None

**install\_app** (*filepath*, *replace=False*)

Install the application on the device

**Parameters**

- **filepath** – full path to the *apk* file to be installed on the device
- **replace** – True or False to replace the existing application

**Returns** output from installation process

**install\_multiple\_app** (*filepath*, *replace=False*)

Install multiple the application on the device

**Parameters**

- **filepath** – full path to the *apk* file to be installed on the device
- **replace** – True or False to replace the existing application

**Returns** output from installation process

**uninstall\_app** (*package*)

Uninstall the application from the device

**Parameters** `package` – package name

**Returns** output from the uninstallation process

**snapshot** (*filename=None*, *ensure\_orientation=True*)

Take the screenshot of the display. The output is send to stdout by default.

**Parameters**

- **filename** – name of the file where to store the screenshot, default is None which si stdout
- **ensure\_orientation** – True or False whether to keep the orientation same as display

**Returns** screenshot output

**shell** (*\*args*, *\*\*kwargs*)

Return *adb shell* interpreter :param *\*args*: optional shell commands :param *\*\*kwargs*: optional shell commands

**Returns** None

**keyevent** (*keyname*, *\*\*kwargs*)

Perform keyevent on the device :param *keyname*: keyeven name :param *\*\*kwargs*: optional arguments

**Returns** None

**wake** ()

Perform wake up event

**Returns** None

**home** ()

Press HOME button

**Returns** None

**text** (*text*, *enter=True*, *\*\*kwargs*)

Input text on the device

**Parameters**

- **text** – text to input
- **enter** – True or False whether to press *Enter* key
- **search** – True or False whether to press *Search* key on IME after input

**Returns** None

**touch** (*pos*, *duration=0.01*)

Perform touch event on the device

**Parameters**

- **pos** – coordinates (x, y)
- **duration** – how long to touch the screen

**Returns** None

**double\_click** (*pos*)

**swipe** (*p1*, *p2*, *duration=0.5*, *steps=5*, *fingers=1*)

Perform swipe event on the device

**Parameters**

- **p1** – start point
- **p2** – end point
- **duration** – how long to swipe the screen, default 0.5
- **steps** – how big is the swipe step, default 5
- **fingers** – the number of fingers. 1 or 2.

**Returns** None

**pinch** (*\*args*, *\*\*kwargs*)

Perform pinch event on the device

**Parameters**

- **\*args** – optional arguments
- **\*\*kwargs** – optional arguments

**Returns** None

**logcat** (*\*args*, *\*\*kwargs*)

Perform `logcat` operations :param *\*args*: optional arguments :param *\*\*kwargs*: optional arguments

**Returns** *logcat* output

**getprop** (*key*, *strip=True*)

Get properties for given key

**Parameters**

- **key** – key name
- **strip** – True or False whether to strip the output or not

**Returns** property value(s)

**get\_ip\_address** ()

Perform several set of commands to obtain the IP address

- *adb shell netcfg | grep wlan0*
- *adb shell ifconfig*
- *adb getprop dhcp.wlan0.ipaddress*

**Returns** None if no IP address has been found, otherwise return the IP address

**get\_top\_activity()**

Get the top activity

**Returns** package, activity and pid

**get\_top\_activity\_name\_and\_pid()**

**get\_top\_activity\_name()**

Get the top activity name

**Returns** package, activity and pid

**is\_keyboard\_shown()**

Return True or False whether soft keyboard is shown or not

### Notes

Might not work on all devices

**Returns** True or False

**is\_screenon()**

Return True or False whether the screen is on or not

### Notes

Might not work on all devices

**Returns** True or False

**is\_locked()**

Return True or False whether the device is locked or not

### Notes

Might not work on some devices

**Returns** True or False

**unlock()**

Unlock the device

### Notes

Might not work on all devices

**Returns** None

**display\_info**

Return the display info (width, height, orientation and max\_x, max\_y)

**Returns** display information

**get\_display\_info()**

Return the display info (width, height, orientation and max\_x, max\_y)

**Returns** display information

**get\_current\_resolution()**

Return current resolution after rotation

**Returns** width and height of the display

**start\_recording(\*args, \*\*kwargs)**

Start recording the device display

**Parameters**

- **\*args** – optional arguments
- **\*\*kwargs** – optional arguments

**Returns** None

**stop\_recording(\*args, \*\*kwargs)**

Stop recording the device display. Recording file will be kept in the device.

**Parameters**

- **\*args** – optional arguments
- **\*\*kwargs** – optional arguments

**Returns** None

### 1.3.1.3 airtest.core.android.constant module

**class CAP\_METHOD**

Bases: object

**MINICAP** = 'MINICAP'

**MINICAP\_STREAM** = 'MINICAP\_STREAM'

**ADBCAP** = 'ADBCAP'

**JAVACAP** = 'JAVACAP'

**class TOUCH\_METHOD**

Bases: object

**MINITOUCH** = 'MINITOUCH'

**ADBT TOUCH** = 'ADBT TOUCH'

**class IME\_METHOD**

Bases: object

**ADBIME** = 'ADBIME'

**YOSEMITEIME** = 'YOSEMITEIME'

**class ORI\_METHOD**

Bases: object

```
ADB = 'ADBORI'
MINICAP = 'MINICAPORI'
```

#### 1.3.1.4 airtest.core.android.ime module

**ensure\_unicode** (*value*)  
Decode UTF-8 values

**Parameters** *value* – value to be decoded

**Returns** decoded valued

**class CustomIme** (*adb, apk\_path, service\_name*)

Bases: object

Input Methods Class Object

**start** ()

Enable input method

**Returns** None

**end** ()

Disable input method

**Returns** None

**text** (*value*)

**class YosemiteIme** (*adb*)

Bases: *airtest.core.android.ime.CustomIme*

Yosemite Input Method Class Object

**start** ()

Enable input method

**Returns** None

**text** (*value*)

Input text with Yosemite input method

**Parameters** *value* – text to be inputted

**Returns** output form *adb shell* command

**code** (*code*)

Sending editor action

**Parameters** *code* – editor action code, e.g., 2 = IME\_ACTION\_GO, 3 = IME\_ACTION\_SEARCH Editor Action Code Ref: <http://developer.android.com/reference/android/view/inputmethod/EditorInfo.html>

**Returns** output form *adb shell* command

#### 1.3.1.5 airtest.core.android.javacap module

**class Javacap** (*adb*)

Bases: *airtest.core.android.yosemite.Yosemite*

This is another screencap class, it is slower in performance than minicap, but it provides the better compatibility

```
APP_PKG = 'com.netease.nie.yosemite'  
SCREENCAP_SERVICE = 'com.netease.nie.yosemite.Capture'  
RECVTIMEOUT = None  
  
get_frames()  
    Get the screen frames  
  
    Returns None  
  
get_frame_from_stream()  
    Get frame from the stream  
  
    Returns frame  
  
teardown_stream()  
    End stream  
  
    Returns None
```

### 1.3.1.6 airtest.core.android.minicap module

```
retry_when_socket_error(func)  
  
class Minicap(adb, projection=None, ori_function=None)  
    Bases: object  
  
    super fast android screenshot method from stf minicap.  
    reference https://github.com/openstf/minicap  
  
    VERSION = 5  
    RECVTIMEOUT = None  
    CMD = 'LD_LIBRARY_PATH=/data/local/tmp /data/local/tmp/minicap'  
  
    install_or_upgrade(*args, **kwargs)  
        Install or upgrade minicap  
  
        Returns None  
  
    uninstall()  
        Uninstall minicap  
  
        Returns None  
  
    install()  
        Install minicap  
  
        Reference: https://github.com/openstf/minicap/blob/master/run.sh  
  
        Returns None  
  
    get_display_info(*args, **kwargs)  
        Get display info by minicap
```

<p><b>Warning:</b> It might segfault, the preferred way is to get the information from adb commands</p>
---

**Returns** display information

```
get_frame(*args, **kwargs)
```

**Get the single frame from minicap -s, this method slower than `get_frames`** 1. shell cmd 1.  
remove log info 1.

**Args:** projection: screenshot projection, default is None which means using self.projection

**Returns:** jpg data

**get\_stream** (\*args, \*\*kwargs)

Get stream, it uses 'adb forward' and socket communication. Use minicap "lazy" mode (provided by gzmaruijie) for long connections - returns one latest frame from the server

**Parameters** **lazy** – True or False

Returns:

**get\_frame\_from\_stream** (\*args, \*\*kwargs)

Get one frame from minicap stream

**Returns** frame

**update\_rotation** (rotation)

Update rotation and reset the backend stream generator

**Parameters** **rotation** – rotation input

**Returns** None

**teardown\_stream** ()

End the stream

**Returns** None

### 1.3.1.7 airtest.core.android.minitouch module

**class Minitouch** (adb, backend=False, ori\_function=None)

Bases: object

Super fast operation from minitouch

References: <https://github.com/openstf/minitouch>

**install\_and\_setup** (\*args, \*\*kwargs)

Install and setup minitouch

**Returns** None

**uninstall** ()

Uninstall minitouch

**Returns** None

**install** ()

Install minitouch

**Returns** None

**setup\_server** ()

Setip minitouch server and adb forward

**Returns** server process

**touch** (\*args, \*\*kwargs)

Perform touch event

minitouch protocol example:

```
d 0 10 10 50
c
<wait in your own code>
u 0
c
```

**Parameters**

- **tuple\_xy** – coordinates (x, y)
- **duration** – time interval for touch event, default is 0.01

**Returns** None

**swipe\_along** (\*args, \*\*kwargs)

Perform swipe event across multiple points in sequence.

**Parameters**

- **coordinates\_list** – list of coordinates: [(x1, y1), (x2, y2), (x3, y3)]
- **duration** – time interval for swipe duration, default is 0.8
- **steps** – size of swipe step, default is 5

**Returns** None

**swipe** (\*args, \*\*kwargs)

Perform swipe event.

**Parameters**

- **tuple\_from\_xy** – start point
- **tuple\_to\_xy** – end point
- **duration** – time interval for swipe duration, default is 0.8
- **steps** – size of swipe step, default is 5

**Returns** None

**two\_finger\_swipe** (\*args, \*\*kwargs)

Perform two finger swipe action

minitouch protocol example:

```
d 0 0 0 50
d 1 1 0 50
c
m 0 20 0 50
m 1 21 0 50
c
m 0 40 0 50
m 1 41 0 50
c
m 0 60 0 50
m 1 61 0 50
```

(continues on next page)



(continued from previous page)

```

c
m 0 80 0 50
m 1 81 0 50
c
m 0 100 0 50
m 1 101 0 50
c
u 0
u 1
c

```

**Parameters**

- **tuple\_from\_xy** – start point
- **tuple\_to\_xy** – end point
- **duration** – time interval for swipe duration, default is 0.8
- **steps** – size of swipe step, default is 5

**Returns** None**pinch** (\*args, \*\*kwargs)

Perform pinch action

minitouch protocol example:

```

d 0 0 100 50
d 1 100 0 50
c
m 0 10 90 50
m 1 90 10 50
c
m 0 20 80 50
m 1 80 20 50
c
m 0 20 80 50
m 1 80 20 50
c
m 0 30 70 50
m 1 70 30 50
c
m 0 40 60 50
m 1 60 40 50
c
m 0 50 50 50
m 1 50 50 50
c
u 0
u 1
c

```

**operate** (\*args, \*\*kwargs)

Perform down, up and move actions

**Parameters** **args** – action arguments, dictionary containing type and x, y coordinates, e.g.:

```
{
  "type" : "down",
  "x" : 10,
  "y" : 10
}
```

**Raises** `RuntimeError` – is invalid arguments are provided

**Returns** `None`

**perform** (*\*args*, *\*\*kwargs*)

Perform a sequence of motion events including: `UpEvent`, `DownEvent`, `MoveEvent`, `SleepEvent` :param `motion_events`: a list of `MotionEvent` instances :param `interval`: minimum interval between events :return: `None`

**safe\_send** (*data*)

Send data to client

**Parameters** `data` – data to send

**Raises** `Exception` – when data cannot be sent

**Returns** `None`

**setup\_client\_backend** ()

Setup backend client thread as daemon

**Returns** `None`

**setup\_client** ()

Setup client in following steps:

```
1. connect to server
2. receive the header
   v <version>
   ^ <max-contacts> <max-x> <max-y> <max-pressure>
   $ <pid>
3. prepare to send
```

**Returns** `None`

**teardown** ()

Stop the server and client

**Returns** `None`

**class MotionEvent**

Bases: `object`

Motion Event to be performed by Minitouch

**getcmd** (*transform=None*)

**class DownEvent** (*coordinates, contact=0, pressure=50*)

Bases: `airtest.core.android.minitouch.MotionEvent`

**getcmd** (*transform=None*)

**class UpEvent** (*contact=0*)

Bases: `airtest.core.android.minitouch.MotionEvent`

**getcmd** (*transform=None*)

```
class MoveEvent (coordinates, contact=0, pressure=50)
    Bases: airtest.core.android.minitouch.MotionEvent

    getcmd (transform=None)
```

```
class SleepEvent (seconds)
    Bases: airtest.core.android.minitouch.MotionEvent

    getcmd (transform=None)
```

### 1.3.1.8 airtest.core.android.recorder module

```
class Recorder (adb)
    Bases: airtest.core.android.yosemite.Yosemite

    Screen recorder

    start_recording (*args, **kwargs)
        Start screen recording

        Parameters

- max_time – maximum rate value, default is 1800
- bit_rate – bit rate value, default is None
- vertical – vertical parameters, default is None

Raises RuntimeError – if any error occurs while setup the recording
Returns None if recording did not start, otherwise True

    stop_recording (*args, **kwargs)
        Stop screen recording

        Parameters

- output – default file is screen.mp4
- is_interrupted – True or False. Stop only, no pulling recorded file from device.

Raises AirtestError – if recording was not started before
Returns None

    pull_last_recording_file (*args, **kwargs)
        Pull the latest recording file from device. Error raises if no recording files on device.

        Parameters output – default file is screen.mp4
```

### 1.3.1.9 airtest.core.android.rotation module

```
class RotationWatcher (adb)
    Bases: object

    RotationWatcher class

    get_ready (*args, **kwargs)

    teardown ()

    start ()
        Start the RotationWatcher daemon thread

        Returns initial orientation
```

**reg\_callback** (*ow\_callback*)

**Parameters** *ow\_callback* –

Returns:

**class XYTransformer**

Bases: object

transform the coordinates (x, y) by orientation (upright <-> original)

**static up\_2\_ori** (*tuple\_xy, tuple\_wh, orientation*)

Transform the coordinates upright -> original

**Parameters**

- **tuple\_xy** – coordinates (x, y)
- **tuple\_wh** – screen width and height
- **orientation** – orientation

**Returns** transformed coordinates (x, y)

**static ori\_2\_up** (*tuple\_xy, tuple\_wh, orientation*)

Transform the coordinates original -> upright

**Parameters**

- **tuple\_xy** – coordinates (x, y)
- **tuple\_wh** – screen width and height
- **orientation** – orientation

**Returns** transformed coordinates (x, y)

### 1.3.1.10 airtest.core.android.yosemite module

**class Yosemite** (*adb*)

Bases: object

Wrapper class of Yosemite.apk, used by javacap/recorder/yosemite\_ime.

**install\_or\_upgrade** ()

Install or update the Yosemite.apk file on the device

**Returns** None

**get\_ready** (*\*args, \*\*kwargs*)

**uninstall** ()

Uninstall *Yosemite.apk* application from the device

**Returns** None

## 1.4 airtest.core.ios package

This package provide IOS Device Class.

## 1.4.1 Submodules

### 1.4.1.1 airtest.core.ios.ios module

**retry\_session** (*func*)

**class IOS** (*addr='http://localhost:8100/'*)

Bases: *airtest.core.device.Device*

ios client # befor this you have to run WebDriverAgent # xcodebuild -project path/to/WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -destination "id=\$(idevice\_id -l)" test # iproxy \$port 8100 \$udid

**uuid**

**session**

**window\_size** (*\*args, \*\*kwargs*)

**orientation**

**display\_info**

**get\_current\_resolution** ()

**home** ()

**snapshot** (*filename=None, strType=False, ensure\_orientation=True*)

take snapshot filename: save screenshot to filename

**touch** (*\*args, \*\*kwargs*)

**double\_click** (*pos*)

**swipe** (*fpos, tpos, duration=0.5, steps=5, fingers=1*)

**keyevent** (*keys*)

just use as home event

**text** (*\*args, \*\*kwargs*)

**install\_app** (*uri, package*)

curl -X POST \$JSON\_HEADER -d '{"desiredCapabilities":{"bundleId":"com.apple.mobilesafari",  
"app":["host\_path]/magicapp.app"}},' \$DEVICE\_URL/session <https://github.com/facebook/WebDriverAgent/wiki/Queries>

**start\_app** (*package, activity=None*)

**stop\_app** (*package*)

**get\_ip\_address** ()

get ip address from webDriverAgent

**Returns** raise if no IP address has been found, otherwise return the IP address

**device\_status** ()

show status return by webDriverAgent Return dicts of infos

### 1.4.1.2 airtest.core.ios.minicap module

**class MinicapIOS** (*udid=None, port=12345*)

Bases: *object*

<https://github.com/openstf/ios-minicap>

```
CAPTIMEOUT = None
setup ()
get_frames ()
    rotation is always right on iOS
list_devices ()
```

## 1.5 airtest.core.win package

### 1.5.1 Submodules

#### 1.5.1.1 airtest.core.win.screen module

#### 1.5.1.2 airtest.core.win.win module

## 1.6 airtest

### 1.6.1 airtest package

#### 1.6.1.1 Subpackages

##### airtest.aircv package

##### Submodules

##### airtest.aircv.aircv module

```
imread (filename)
    cv2.
imwrite (filename, img)
show (img, title='show_img', test_flag=False)
.
show_origin_size (img, title='image', test_flag=False)
.
rotate (img, angle=90, clockwise=True)
    90180270. clockwise=True
crop_image (img, rect)
    ; Crop image , rect = [x_min, y_min, x_max ,y_max]. (airtest)
mark_point (img, point)
    :
mask_image (img, mask, color=(255, 255, 255), linewidth=-1)
    screenmaskgbr(255, 255, 255). mask: [x_min, y_min, x_max, y_max]. color: blue-green-red. linewidth: -1.
get_resolution (img)
```

## airtest.aircv.cal\_confidence module

These functions calculate the similarity of two images of the same size.

**cal\_ccoeff\_confidence** (*im\_source, im\_search*)  
TM\_CCOEFF\_NORMED.

**cal\_rgb\_confidence** (*img\_src\_rgb, img\_sch\_rgb*)

## airtest.aircv.error module

**Declaration:** Define all BaseError Classes used in aircv.

**exception BaseError** (*message=""*)  
Bases: `exceptions.Exception`

Base class for exceptions in this module.

**exception FileNotExistError** (*message=""*)  
Bases: `airtest.aircv.error.BaseError`

Image does not exist.

**exception TemplateInputError** (*message=""*)  
Bases: `airtest.aircv.error.BaseError`

Resolution input is not right.

**exception NoSIFTModuleError** (*message=""*)  
Bases: `airtest.aircv.error.BaseError`

Resolution input is not right.

**exception NoSiftMatchPointError** (*message=""*)  
Bases: `airtest.aircv.error.BaseError`

Exception raised for errors 0 sift points found in the input images.

**exception SiftResultCheckError** (*message=""*)  
Bases: `airtest.aircv.error.BaseError`

Exception raised for errors 0 sift points found in the input images.

**exception HomographyError** (*message=""*)  
Bases: `airtest.aircv.error.BaseError`

In homography, find no mask, should kill points which is duplicate.

## airtest.aircv.sift module

**find\_sift** (*im\_source, im\_search, threshold=0.8, rgb=True, good\_ratio=0.59*)  
sift.

**mask\_sift** (*im\_source, im\_search, threshold=0.8, rgb=True, good\_ratio=0.59*)  
sift.

**find\_all\_sift** (*im\_source, im\_search, threshold=0.8, rgb=True, good\_ratio=0.59*)  
sift.

### airtest.aircv.template module

[summary] :

[description]

1. threshod: 0.8
2. rgb: ,.

**find\_template** (*im\_source, im\_search, threshold=0.8, rgb=False*)

**find\_all\_template** (*im\_source, im\_search, threshold=0.8, rgb=False, max\_count=10*)

### airtest.aircv.utils module

**generate\_result** (*middle\_point, pypts, confi*)

Format the result: .

**check\_image\_valid** (*im\_source, im\_search*)

Check if the input images valid or not.

**check\_source\_larger\_than\_search** (*im\_source, im\_search*)

**img\_mat\_rgb\_2\_gray** (*img\_mat*)

Turn img\_mat into gray\_scale, so that template match can figure the img data. “print(type(im\_search[0][0]))” can check the pixel type.

**img\_2\_string** (*img*)

**string\_2\_img** (*pngstr*)

**pil\_2\_cv2** (*pil\_image*)

**cv2\_2\_pil** (*cv2\_image*)

### airtest.cli package

#### Submodules

#### airtest.cli.info module

**get\_script\_info** (*script\_path*)

extract info from script, like basename, \_\_author\_\_, \_\_title\_\_ and \_\_desc\_\_.

**get\_author\_title\_desc** (*text*)

Get author title desc.

**process\_desc** (*desc*)

**strip\_str** (*string*)

Strip string.



## airtest.cli.parser module

**get\_parser** ()

**runner\_parser** (*ap=None*)

**cli\_setup** (*args=None*)  
future api for setup env by cli

## airtest.cli.runner module

**class AirtestCase** (*methodName='runTest'*)

Bases: `unittest.case.TestCase`

**PROJECT\_ROOT** = '.'

**SCRIPTTEXT** = '.air'

**TPLEXT** = '.png'

**classmethod setUpClass** ()

Hook method for setting up class fixture before running tests in the class.

**setUp** ()

Hook method for setting up the test fixture before exercising it.

**tearDown** ()

Hook method for deconstructing the test fixture after testing it.

**runTest** ()

**classmethod exec\_other\_script** (*scriptpath*)

run other script in test script

**setup\_by\_args** (*args*)

**run\_script** (*parsed\_args, testcase\_cls=<class 'airtest.cli.runner.AirtestCase'>*)

## airtest.core package

### Subpackages

### Submodules

## airtest.core.cv module

“Airtest.

**loop\_find** (*\*args, \*\*kwargs*)

Search for image template in the screen until timeout

### Parameters

- **query** – image template to be found in screenshot
- **timeout** – time interval how long to look for the image template
- **threshold** – default is None
- **interval** – sleep interval before next attempt to find the image template

- **intervalfunc** – function that is executed after unsuccessful attempt to find the image template

**Raises** `TargetNotFoundError` – when image template is not found in screenshot

**Returns** `TargetNotFoundError` if image template not found, otherwise returns the position where the image template has been found in screenshot

**try\_log\_screen** (*\*args, \*\*kwargs*)

Save screenshot to file

**Parameters** **screen** – screenshot to be saved

**Returns** `None`

**class Template** (*filename, threshold=None, target\_pos=5, record\_pos=None, resolution=(), rgb=False*)

Bases: `object`

picture as touch/swipe/wait/exists target and extra info for cv match filename: pic filename target\_pos: ret which pos in the pic record\_pos: pos in screen when recording resolution: screen resolution when recording rgb: rgb.

**filepath**

**match\_in** (*screen*)

**match\_all\_in** (*screen*)

**class Predictor**

Bases: `object`

this class predicts the `press_point` and the area to search `im_search`.

**DEVIATION** = 100

**static count\_record\_pos** (*pos, resolution*)

.

**classmethod get\_predict\_point** (*record\_pos, screen\_resolution*)

.

**classmethod get\_predict\_area** (*record\_pos, image\_wh, image\_resolution=(), screen\_resolution=()*)

Get predicted area in screen.

### airtest.core.device module

**class MetaDevice**

Bases: `type`

**REGISTRY** = {'Android': <class 'airtest.core.android.android.Android'>, 'Device': <cl

**class Device**

Bases: `object`

base class for test device

**uuid**

**shell** (*\*args, \*\*kwargs*)

**snapshot** (*\*args, \*\*kwargs*)

**touch** (*target, \*\*kwargs*)

**double\_click** (*target*)

```

swipe (t1, t2, **kwargs)
keyevent (key, **kwargs)
text (text, enter=True)
start_app (package, **kwargs)
stop_app (package)
clear_app (package)
list_app (**kwargs)
install_app (uri, **kwargs)
uninstall_app (package)
get_current_resolution ()
get_ip_address ()

```

## airtest.core.error module

error classes

```

exception BaseError (value)
    Bases: exceptions.Exception

```

```

exception AirstestError (value)
    Bases: airtest.core.error.BaseError

    This is Airstest BaseError

```

```

exception TargetNotFoundError (value)
    Bases: airtest.core.error.AirstestError

    This is TargetNotFoundError BaseError When something is not found

```

```

exception ScriptParamError (value)
    Bases: airtest.core.error.AirstestError

    This is ScriptParamError BaseError When something goes wrong

```

```

exception AdbError (stdout, stderr)
    Bases: exceptions.Exception

    This is AdbError BaseError When ADB have something wrong

```

```

exception AdbShellError (stdout, stderr)
    Bases: airtest.core.error.AdbError

    adb shell error

```

```

exception DeviceConnectionError (value)
    Bases: airtest.core.error.BaseError

    device connection error

```

```

    DEVICE_CONNECTION_ERROR = "error:\s*((device '\\\\S+\\\' not found)|(cannot connect to

```

```

exception ICmdError (stdout, stderr)
    Bases: exceptions.Exception

    This is ICmdError BaseError When ICmd have something wrong

```

**exception MinicapError** (*value*)

Bases: `airtest.core.error.BaseError`

This is MinicapError BaseError When Minicap have something wrong

**exception MinitouchError** (*value*)

Bases: `airtest.core.error.BaseError`

This is MinitouchError BaseError When Minicap have something wrong

**exception PerformanceError** (*value*)

Bases: `airtest.core.error.BaseError`

### airtest.core.helper module

**class G**

Bases: `object`

Represent the globals variables

**BASEDIR** = []

**LOGGER** = `<airtest.utils.logwraper.AirtestLogger object>`

**LOGGING** = `<logging.Logger object>`

**SCREEN** = `None`

**DEVICE** = `None`

**DEVICE\_LIST** = []

**RECENT\_CAPTURE** = `None`

**RECENT\_CAPTURE\_PATH** = `None`

**CUSTOM\_DEVICES** = {}

**classmethod add\_device** (*dev*)

Add device instance in G and set as current device.

#### Examples

```
G.add_device(Android())
```

**Parameters** *dev* – device to init

**Returns** `None`

**classmethod register\_custom\_device** (*device\_cls*)

**set\_logdir** (*dirpath*)

set log dir for logfile and screenshots.

**Parameters** *dirpath* – directory to save logfile and screenshots

Returns:

**log** (*message*, *traceback=""*)

Insert user log, will be displayed in Html report.

**Parameters**

- **message** – log message

- **traceback** – log traceback if exists, use `traceback.format_exc` to get best format

**Returns** None

```
logwrap(f)
device_platform(device=None)
using(path)
import_device_cls(platform)
    lazy import device class
delay_after_operation()
```

### airtest.core.settings module

```
class Settings
    Bases: object
    DEBUG = False
    LOG_DIR = None
    LOG_FILE = 'log.txt'
    static RESIZE_METHOD(w, h, sch_resolution, src_resolution, design_resolution=(960, 640))
        : COCOSMIN.
    CVSTRATEGY = ['tpl', 'sift']
    THRESHOLD = 0.7
    THRESHOLD_STRICT = 0.7
    OPDELAY = 0.1
    FIND_TIMEOUT = 20
    FIND_TIMEOUT_TMP = 3
    PROJECT_ROOT = ''
```

### airtest.report package

#### Submodules

#### airtest.report.report module

```
n12br(eval_ctx, value)
class LogToHtml(script_root, log_root="", static_root="", export_dir=None, script_name="", log-
    file='log.txt', lang='en', plugins=None)
    Bases: object
    Convert log to html display
    scale = 0.5
    static init_plugin_modules(plugins)
```

```
static div_rect (r)  
    count rect for js use  
is_pos (v)  
copy_tree (src, dst, ignore=None)  
report (template_name, output_file=None, record_list=None)  
simple_report (filepath, logpath=True, logfile='log.txt', output='log.html')  
get_parger (ap)  
main (args)
```

### a

- airtest, 34
- airtest.aircv, 34
- airtest.aircv.aircv, 34
- airtest.aircv.cal\_confidence, 35
- airtest.aircv.error, 35
- airtest.aircv.sift, 35
- airtest.aircv.template, 36
- airtest.aircv.utils, 36
- airtest.cli, 36
- airtest.cli.info, 36
- airtest.cli.parser, 37
- airtest.cli.runner, 37
- airtest.core, 37
- airtest.core.android, 12
- airtest.core.android.adb, 12
- airtest.core.android.android, 20
- airtest.core.android.constant, 24
- airtest.core.android.ime, 25
- airtest.core.android.javacap, 25
- airtest.core.android.minicap, 26
- airtest.core.android.minitouch, 27
- airtest.core.android.recorder, 31
- airtest.core.android.rotation, 31
- airtest.core.android.yosemite, 32
- airtest.core.api, 6
- airtest.core.cv, 37
- airtest.core.device, 38
- airtest.core.error, 39
- airtest.core.helper, 40
- airtest.core.ios, 32
- airtest.core.ios.ios, 33
- airtest.core.ios.minicap, 33
- airtest.core.settings, 41
- airtest.report, 41
- airtest.report.report, 41





## A

ADB (class in `airtest.core.android.adb`), 12  
 ADB (ORI\_METHOD attribute), 24  
 ADBCAP (CAP\_METHOD attribute), 24  
 AdbError, 39  
 ADBIME (IME\_METHOD attribute), 24  
 AdbShellError, 39  
 ADBTOUCH (TOUCH\_METHOD attribute), 24  
 add\_device() (`airtest.core.helper.G` class method), 40  
 airtest (module), 34  
 airtest.aircv (module), 34  
 airtest.aircv.aircv (module), 34  
 airtest.aircv.cal\_confidence (module), 35  
 airtest.aircv.error (module), 35  
 airtest.aircv.sift (module), 35  
 airtest.aircv.template (module), 36  
 airtest.aircv.utils (module), 36  
 airtest.cli (module), 36  
 airtest.cli.info (module), 36  
 airtest.cli.parser (module), 37  
 airtest.cli.runner (module), 37  
 airtest.core (module), 37  
 airtest.core.android (module), 12  
 airtest.core.android.adb (module), 12  
 airtest.core.android.android (module), 20  
 airtest.core.android.constant (module), 24  
 airtest.core.android.ime (module), 25  
 airtest.core.android.javacap (module), 25  
 airtest.core.android.minicap (module), 26  
 airtest.core.android.minitouch (module), 27  
 airtest.core.android.recorder (module), 31  
 airtest.core.android.rotation (module), 31  
 airtest.core.android.yosemite (module), 32  
 airtest.core.api (module), 6  
 airtest.core.cv (module), 37  
 airtest.core.device (module), 38  
 airtest.core.error (module), 39  
 airtest.core.helper (module), 40  
 airtest.core.ios (module), 32

airtest.core.ios.ios (module), 33  
 airtest.core.ios.minicap (module), 33  
 airtest.core.settings (module), 41  
 airtest.report (module), 41  
 airtest.report.report (module), 41  
 AirtestCase (class in `airtest.cli.runner`), 37  
 AirtestError, 39  
 Android (class in `airtest.core.android.android`), 20  
 APP\_PKG (Javacap attribute), 25  
 assert\_equal() (in module `airtest.core.api`), 11  
 assert\_exists() (in module `airtest.core.api`), 11  
 assert\_not\_equal() (in module `airtest.core.api`), 11  
 assert\_not\_exists() (in module `airtest.core.api`), 11  
 auto\_setup() (in module `airtest.core.api`), 7

## B

BASEDIR (G attribute), 40  
 BaseError, 35, 39  
 builtin\_adb\_path() (ADB static method), 12

## C

cal\_ccoeff\_confidence() (in module `airtest.aircv.cal_confidence`), 35  
 cal\_rgb\_confidence() (in module `airtest.aircv.cal_confidence`), 35  
 CAP\_METHOD (class in `airtest.core.android.constant`), 24  
 CAPTIMEOUT (MinicapIOS attribute), 33  
 check\_app() (ADB method), 18  
 check\_app() (Android method), 20  
 check\_image\_valid() (in module `airtest.aircv.utils`), 36  
 check\_source\_larger\_than\_search() (in module `airtest.aircv.utils`), 36  
 cleanup\_adb\_forward() (in module `airtest.core.android.adb`), 19  
 clear\_app() (ADB method), 19  
 clear\_app() (Android method), 20  
 clear\_app() (Device method), 39  
 clear\_app() (in module `airtest.core.api`), 8

cli\_setup() (in module airtest.cli.parser), 37  
 click() (in module airtest.core.api), 9  
 close\_proc\_pipe() (ADB method), 13  
 CMD (Minicap attribute), 26  
 cmd() (ADB method), 12  
 code() (YosemiteIme method), 25  
 connect() (ADB method), 13  
 connect\_device() (in module airtest.core.api), 7  
 copy\_tree() (LogToHtml method), 42  
 count\_record\_pos() (Predictor static method), 38  
 crop\_image() (in module airtest.aircv.aircv), 34  
 CUSTOM\_DEVICES (G attribute), 40  
 CustomIme (class in airtest.core.android.ime), 25  
 cv2\_2\_pil() (in module airtest.aircv.utils), 36  
 CVSTRATEGY (Settings attribute), 41

## D

DEBUG (Settings attribute), 41  
 delay\_after\_operation() (in module airtest.core.helper), 41  
 DEVIATION (Predictor attribute), 38  
 Device (class in airtest.core.device), 38  
 DEVICE (G attribute), 40  
 device() (in module airtest.core.api), 7  
 DEVICE\_CONNECTION\_ERROR (DeviceConnectionError attribute), 39  
 DEVICE\_LIST (G attribute), 40  
 device\_platform() (in module airtest.core.helper), 41  
 device\_status() (IOS method), 33  
 DeviceConnectionError, 39  
 devices() (ADB method), 13  
 disconnect() (ADB method), 13  
 display\_info (ADB attribute), 16  
 display\_info (Android attribute), 23  
 display\_info (IOS attribute), 33  
 div\_rect() (LogToHtml static method), 41  
 double\_click() (Android method), 22  
 double\_click() (Device method), 38  
 double\_click() (in module airtest.core.api), 9  
 double\_click() (IOS method), 33  
 DownEvent (class in airtest.core.android.minitouch), 30

## E

end() (CustomIme method), 25  
 ensure\_unicode() (in module airtest.core.android.ime), 25  
 exec\_other\_script() (airtest.cli.runner.AirtestCase class method), 37  
 exists() (in module airtest.core.api), 10  
 exists\_file() (ADB method), 16

## F

file\_size() (ADB method), 16  
 FileNotExistError, 35  
 filepath (Template attribute), 38

find\_all() (in module airtest.core.api), 10  
 find\_all\_sift() (in module airtest.aircv.sift), 35  
 find\_all\_template() (in module airtest.aircv.template), 36  
 find\_sift() (in module airtest.aircv.sift), 35  
 find\_template() (in module airtest.aircv.template), 36  
 FIND\_TIMEOUT (Settings attribute), 41  
 FIND\_TIMEOUT\_TMP (Settings attribute), 41  
 forward() (ADB method), 14

## G

G (class in airtest.core.helper), 40  
 generate\_result() (in module airtest.aircv.utils), 36  
 get\_author\_title\_desc() (in module airtest.cli.info), 36  
 get\_available\_forward\_local() (airtest.core.android.adb.ADB class method), 15  
 get\_cpufreq() (ADB method), 19  
 get\_cpuinfo() (ADB method), 19  
 get\_current\_resolution() (Android method), 24  
 get\_current\_resolution() (Device method), 39  
 get\_current\_resolution() (IOS method), 33  
 get\_default\_device() (Android method), 20  
 get\_device\_info() (ADB method), 19  
 get\_display\_info() (ADB method), 17  
 get\_display\_info() (Android method), 24  
 get\_display\_info() (Minicap method), 26  
 get\_forwards() (ADB method), 14  
 get\_frame() (Minicap method), 26  
 get\_frame\_from\_stream() (Javacap method), 26  
 get\_frame\_from\_stream() (Minicap method), 27  
 get\_frames() (Javacap method), 26  
 get\_frames() (MinicapIOS method), 34  
 get\_gateway\_address() (ADB method), 19  
 get\_gpu() (ADB method), 19  
 get\_ip\_address() (ADB method), 19  
 get\_ip\_address() (Android method), 22  
 get\_ip\_address() (Device method), 39  
 get\_ip\_address() (IOS method), 33  
 get\_manufacturer() (ADB method), 19  
 get\_memory() (ADB method), 19  
 get\_model() (ADB method), 19  
 get\_package\_version() (ADB method), 18  
 get\_parger() (in module airtest.report.report), 42  
 get\_parser() (in module airtest.cli.parser), 37  
 get\_predict\_area() (airtest.core.cv.Predictor class method), 38  
 get\_predict\_point() (airtest.core.cv.Predictor class method), 38  
 get\_ready() (RotationWatcher method), 31  
 get\_ready() (Yosemite method), 32  
 get\_resolution() (in module airtest.aircv.aircv), 34  
 get\_script\_info() (in module airtest.cli.info), 36  
 get\_status() (ADB method), 13  
 get\_storage() (ADB method), 19

[get\\_stream\(\)](#) (Minicap method), 27  
[get\\_top\\_activity\(\)](#) (ADB method), 17  
[get\\_top\\_activity\(\)](#) (Android method), 23  
[get\\_top\\_activity\\_name\(\)](#) (Android method), 23  
[get\\_top\\_activity\\_name\\_and\\_pid\(\)](#) (Android method), 23  
[getcmd\(\)](#) (DownEvent method), 30  
[getcmd\(\)](#) (MotionEvent method), 30  
[getcmd\(\)](#) (MoveEvent method), 31  
[getcmd\(\)](#) (SleepEvent method), 31  
[getcmd\(\)](#) (UpEvent method), 30  
[getDisplayOrientation\(\)](#) (ADB method), 17  
[getMaxXY\(\)](#) (ADB method), 17  
[getPhysicalDisplayInfo\(\)](#) (ADB method), 17  
[getprop\(\)](#) (ADB method), 14  
[getprop\(\)](#) (Android method), 22  
[getRestrictedScreen\(\)](#) (ADB method), 17

## H

[home\(\)](#) (Android method), 21  
[home\(\)](#) (in module `airtest.core.api`), 8  
[home\(\)](#) (IOS method), 33  
[HomographyError](#), 35

## I

[ICmdError](#), 39  
[IME\\_METHOD](#) (class in `airtest.core.android.constant`), 24  
[img\\_2\\_string\(\)](#) (in module `airtest.aircv.utils`), 36  
[img\\_mat\\_rgb\\_2\\_gray\(\)](#) (in module `airtest.aircv.utils`), 36  
[import\\_device\\_cls\(\)](#) (in module `airtest.core.helper`), 41  
[imread\(\)](#) (in module `airtest.aircv.aircv`), 34  
[imwrite\(\)](#) (in module `airtest.aircv.aircv`), 34  
[init\\_device\(\)](#) (in module `airtest.core.api`), 6  
[init\\_plugin\\_modules\(\)](#) (`LogToHtml` static method), 41  
[install\(\)](#) (in module `airtest.core.api`), 8  
[install\(\)](#) (Minicap method), 26  
[install\(\)](#) (Minitouch method), 27  
[install\\_and\\_setup\(\)](#) (Minitouch method), 27  
[install\\_app\(\)](#) (ADB method), 15  
[install\\_app\(\)](#) (Android method), 21  
[install\\_app\(\)](#) (Device method), 39  
[install\\_app\(\)](#) (IOS method), 33  
[install\\_multiple\\_app\(\)](#) (ADB method), 15  
[install\\_multiple\\_app\(\)](#) (Android method), 21  
[install\\_or\\_upgrade\(\)](#) (Minicap method), 26  
[install\\_or\\_upgrade\(\)](#) (Yosemite method), 32  
[IOS](#) (class in `airtest.core.ios.ios`), 33  
[is\\_keyboard\\_shown\(\)](#) (ADB method), 17  
[is\\_keyboard\\_shown\(\)](#) (Android method), 23  
[is\\_locked\(\)](#) (ADB method), 17  
[is\\_locked\(\)](#) (Android method), 23  
[is\\_pos\(\)](#) (`LogToHtml` method), 42  
[is\\_screenon\(\)](#) (ADB method), 17  
[is\\_screenon\(\)](#) (Android method), 23

## J

[JAVACAP](#) (`CAP_METHOD` attribute), 24  
[Javacap](#) (class in `airtest.core.android.javacap`), 25

## K

[keyevent\(\)](#) (ADB method), 14  
[keyevent\(\)](#) (Android method), 21  
[keyevent\(\)](#) (Device method), 39  
[keyevent\(\)](#) (in module `airtest.core.api`), 10  
[keyevent\(\)](#) (IOS method), 33  
[kill\\_server\(\)](#) (ADB method), 12

## L

[line\\_breaker](#) (ADB attribute), 16  
[list\\_app\(\)](#) (ADB method), 18  
[list\\_app\(\)](#) (Android method), 20  
[list\\_app\(\)](#) (Device method), 39  
[list\\_devices\(\)](#) (in module `airtest.core.ios.minicap`), 34  
[log\(\)](#) (in module `airtest.core.helper`), 40  
[LOG\\_DIR](#) (Settings attribute), 41  
[LOG\\_FILE](#) (Settings attribute), 41  
[logcat\(\)](#) (ADB method), 16  
[logcat\(\)](#) (Android method), 22  
[LOGGER](#) (G attribute), 40  
[LOGGING](#) (G attribute), 40  
[LogToHtml](#) (class in `airtest.report.report`), 41  
[logwrap\(\)](#) (in module `airtest.core.helper`), 41  
[loop\\_find\(\)](#) (in module `airtest.core.cv`), 37

## M

[main\(\)](#) (in module `airtest.report.report`), 42  
[mark\\_point\(\)](#) (in module `airtest.aircv.aircv`), 34  
[mask\\_image\(\)](#) (in module `airtest.aircv.aircv`), 34  
[mask\\_sift\(\)](#) (in module `airtest.aircv.sift`), 35  
[match\\_all\\_in\(\)](#) (Template method), 38  
[match\\_in\(\)](#) (Template method), 38  
[MetaDevice](#) (class in `airtest.core.device`), 38  
[MINICAP](#) (`CAP_METHOD` attribute), 24  
[Minicap](#) (class in `airtest.core.android.minicap`), 26  
[MINICAP](#) (`ORI_METHOD` attribute), 25  
[MINICAP\\_STREAM](#) (`CAP_METHOD` attribute), 24  
[MinicapError](#), 39  
[MinicapIOS](#) (class in `airtest.core.ios.minicap`), 33  
[Minitouch](#) (class in `airtest.core.android.minitouch`), 27  
[MINITOUCH](#) (`TOUCH_METHOD` attribute), 24  
[MinitouchError](#), 40  
[MotionEvent](#) (class in `airtest.core.android.minitouch`), 30  
[MoveEvent](#) (class in `airtest.core.android.minitouch`), 30

## N

[nl2br\(\)](#) (in module `airtest.report.report`), 41  
[NoSiftMatchPointError](#), 35  
[NoSIFTModuleError](#), 35

## O

OPDELAY (Settings attribute), 41  
 operate() (Minitouch method), 29  
 ori\_2\_up() (XYTransformer static method), 32  
 ORI\_METHOD (class in airtest.core.android.constant), 24  
 orientation (IOS attribute), 33

## P

path\_app() (ADB method), 18  
 path\_app() (Android method), 20  
 perform() (Minitouch method), 30  
 PerformanceError, 40  
 pil\_2\_cv2() (in module airtest.aircv.utils), 36  
 pinch() (Android method), 22  
 pinch() (in module airtest.core.api), 9  
 pinch() (Minitouch method), 29  
 pm\_install() (ADB method), 15  
 pm\_uninstall() (ADB method), 15  
 Predictor (class in airtest.core.cv), 38  
 process\_desc() (in module airtest.cli.info), 36  
 PROJECT\_ROOT (AirtestCase attribute), 37  
 PROJECT\_ROOT (Settings attribute), 41  
 pull() (ADB method), 14  
 pull\_last\_recording\_file() (Recorder method), 31  
 push() (ADB method), 14

## R

raw\_shell() (ADB method), 13  
 RECENT\_CAPTURE (G attribute), 40  
 RECENT\_CAPTURE\_PATH (G attribute), 40  
 Recorder (class in airtest.core.android.recorder), 31  
 RECVMETHOD (Javacap attribute), 26  
 RECVMETHOD (Minicap attribute), 26  
 reg\_callback() (RotationWatcher method), 32  
 register\_custom\_device() (airtest.core.helper.G class method), 40  
 REGISTRY (MetaDevice attribute), 38  
 remove\_forward() (ADB method), 15  
 report() (LogToHtml method), 42  
 RESIZE\_METHOD() (Settings static method), 41  
 retry\_session() (in module airtest.core.ios.ios), 33  
 retry\_when\_socket\_error() (in module airtest.core.android.minicap), 26  
 rotate() (in module airtest.aircv.aircv), 34  
 RotationWatcher (class in airtest.core.android.rotation), 31  
 run\_script() (in module airtest.cli.runner), 37  
 runner\_parser() (in module airtest.cli.parser), 37  
 runTest() (AirtestCase method), 37

## S

safe\_send() (Minitouch method), 30

scale (LogToHtml attribute), 41  
 SCREEN (G attribute), 40  
 SCREENCAP\_SERVICE (Javacap attribute), 26  
 SCRIPTTEXT (AirtestCase attribute), 37  
 ScriptParamError, 39  
 sdk\_version (ADB attribute), 14  
 session (IOS attribute), 33  
 set\_current() (in module airtest.core.api), 7  
 set\_logdir() (in module airtest.core.helper), 40  
 Settings (class in airtest.core.settings), 41  
 setUp() (AirtestCase method), 37  
 setup() (MinicapIOS method), 34  
 setup\_by\_args() (in module airtest.cli.runner), 37  
 setup\_client() (Minitouch method), 30  
 setup\_client\_backend() (Minitouch method), 30  
 setup\_forward() (ADB method), 15  
 setup\_server() (Minitouch method), 27  
 setUpClass() (airtest.cli.runner.AirtestCase class method), 37  
 shell() (ADB method), 13  
 shell() (Android method), 21  
 shell() (Device method), 38  
 shell() (in module airtest.core.api), 7  
 SHELL\_ENCODING (ADB attribute), 12  
 show() (in module airtest.aircv.aircv), 34  
 show\_origin\_size() (in module airtest.aircv.aircv), 34  
 SiftResultCheckError, 35  
 simple\_report() (in module airtest.report.report), 42  
 sleep() (in module airtest.core.api), 10  
 SleepEvent (class in airtest.core.android.minitouch), 31  
 snapshot() (ADB method), 16  
 snapshot() (Android method), 21  
 snapshot() (Device method), 38  
 snapshot() (in module airtest.core.api), 8  
 snapshot() (IOS method), 33  
 start() (CustomIme method), 25  
 start() (RotationWatcher method), 31  
 start() (YosemiteIme method), 25  
 start\_app() (ADB method), 18  
 start\_app() (Android method), 20  
 start\_app() (Device method), 39  
 start\_app() (in module airtest.core.api), 7  
 start\_app() (IOS method), 33  
 start\_app\_timing() (ADB method), 18  
 start\_app\_timing() (Android method), 20  
 start\_cmd() (ADB method), 12  
 start\_recording() (Android method), 24  
 start\_recording() (Recorder method), 31  
 start\_server() (ADB method), 12  
 start\_shell() (ADB method), 13  
 status\_device (ADB attribute), 12  
 status\_offline (ADB attribute), 12  
 stop\_app() (ADB method), 19  
 stop\_app() (Android method), 20

stop\_app() (Device method), 39  
 stop\_app() (in module airtest.core.api), 7  
 stop\_app() (IOS method), 33  
 stop\_recording() (Android method), 24  
 stop\_recording() (Recorder method), 31  
 string\_2\_img() (in module airtest.aircv.utils), 36  
 strip\_str() (in module airtest.cli.info), 36  
 swipe() (ADB method), 16  
 swipe() (Android method), 22  
 swipe() (Device method), 38  
 swipe() (in module airtest.core.api), 9  
 swipe() (IOS method), 33  
 swipe() (Minitouch method), 28  
 swipe\_along() (Minitouch method), 28

## T

TargetNotFoundError, 39  
 tearDown() (AirtestCase method), 37  
 teardown() (Minitouch method), 30  
 teardown() (RotationWatcher method), 31  
 teardown\_stream() (Javacap method), 26  
 teardown\_stream() (Minicap method), 27  
 Template (class in airtest.core.cv), 38  
 TemplateInputError, 35  
 text() (Android method), 21  
 text() (CustomIme method), 25  
 text() (Device method), 39  
 text() (in module airtest.core.api), 10  
 text() (IOS method), 33  
 text() (YosemiteIme method), 25  
 THRESHOLD (Settings attribute), 41  
 THRESHOLD\_STRICT (Settings attribute), 41  
 touch() (ADB method), 16  
 touch() (Android method), 22  
 touch() (Device method), 38  
 touch() (in module airtest.core.api), 8  
 touch() (IOS method), 33  
 touch() (Minitouch method), 27  
 TOUCH\_METHOD (class in airtest.core.android.constant), 24  
 TPLEXT (AirtestCase attribute), 37  
 try\_log\_screen() (in module airtest.core.cv), 38  
 two\_finger\_swipe() (Minitouch method), 28

## U

uninstall() (in module airtest.core.api), 8  
 uninstall() (Minicap method), 26  
 uninstall() (Minitouch method), 27  
 uninstall() (Yosemite method), 32  
 uninstall\_app() (ADB method), 15  
 uninstall\_app() (Android method), 21  
 uninstall\_app() (Device method), 39  
 unlock() (ADB method), 17  
 unlock() (Android method), 23

up\_2\_ori() (XYTransformer static method), 32  
 update\_rotation() (Minicap method), 27  
 UpEvent (class in airtest.core.android.minitouch), 30  
 using() (in module airtest.core.helper), 41  
 uuid (Android attribute), 20  
 uuid (Device attribute), 38  
 uuid (IOS attribute), 33

## V

VERSION (Minicap attribute), 26  
 version() (ADB method), 12

## W

wait() (in module airtest.core.api), 10  
 wait\_for\_device() (ADB method), 13  
 wake() (Android method), 21  
 wake() (in module airtest.core.api), 8  
 window\_size() (IOS method), 33

## X

XYTransformer (class in airtest.core.android.rotation), 32

## Y

Yosemite (class in airtest.core.android.yosemite), 32  
 YosemiteIme (class in airtest.core.android.ime), 25  
 YOSEMITEIME (IME\_METHOD attribute), 24

in