airtest Documentation

Release

Game-Netease

Apr 12, 2018

Quickstart

CHAPTER 1

Airtest

UI Test Automation Framework for Games and Apps

Airtest is a cross-platform automated testing framework with main focus on games, which can also be used for native apps. Currently, Windows and Android are well supported. Support for iOS comes in near future.

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

UI Test Automation Framework for Games and Apps

1.1.1 Getting Started

Airtest is a cross-platform automated testing framework focusing mainly on games, but can also be used for native apps. Windows and Android are currently supported; iOS support will be available in the near future.

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 test on games without injecting any code. The test will generate an HTML report, which allows you to quickly locate failed test cases.

AirtestIDE is an out of the box GUI tool that helps to create and record test cases in a user-friendly way. AirtestIDE provides QA with a complete production workflow: `record -> replay -> report`

Get Started from Airtest Project Homepage

1.1.2 Installation

This section describes how to install Airtest test framework. Download AirtestIDE from our homepage if you need to use the GUI tool.

System Requirements

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

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 simple API-specific test scenario 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
connect_device("Android:///")
install("path/to/your/apk")
start_app("package_name_of_your_apk")
touch("image_of_a_button.png")
swipe("slide_start.png", "slide_end.png")
assert_exists("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 APIs, so that you can write test once and run test on different devices.

1. Using connect_device API you can connect to any android device or windows application.
2. Then perform simulated input to test your game or app.
3. And **do not** forget to make assertions of the expected test result.

Connect Device

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

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

Connect android device

Local 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 a local adb device using 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 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.

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.

Make Assertion

Airtest provide some assert functions, including:

- assert_exists
- assert_not_exists
- assert_equal
- assert_not_equal

When assertion fails, it will raise `AssertsionError`. 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 tests as `.air` directories. Airtest CLI provides the possibility to execute tests on different host machine and target device platforms without using AirtestIDE itself.

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

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

run test case

```
# run test test 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
```

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
```

get test 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.

connect_device(*uri*)
Initialize device with uri and set the device as the current one.

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 adb device using custom params
- `windows:///` # local Windows application
- `ios:///` # iOS device

Platforms Android, iOS, Windows

device()
Return the current active device.

Returns current device instance

Platforms Android, iOS, Windows

set_current(**args*, ***kwargs*)
Set current active device.

Parameters **index** – index of initialized device instance

Raises **IndexError** – raised when device index is out of device list

Returns None

Platforms Android, iOS, Windows

auto_setup(*filepath*)
Auto setup G.BASEDIR and try connect android device if not connected to any device.

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)
- **kwargs** – platform specific *kwargs*, please refer to corresponding docs

Returns None

Platforms Android, Windows, iOS

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 None

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

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)

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 od 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 propery 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(**args*, ***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

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 exits on the device

Parameters **filepath** – path to the file

Returns True or False if file found or not

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

getPhysicalDisplayInfo()

Display size and resolution to be obtained:

1. physical display size (physical_width, physical_height) of the device - this is used by *minitouch* as a coordinates system

1. screen effective resolution (width, height) - this is used by game image adaptation as a coordinates system

1. click range resolution (max_x, max_y)

Returns display size and resolution information

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

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

cleanup_adb_forward()

airtest.core.android.android module

class Android(*serialno=None*, *host=None*, *cap_method='MINICAP_STREAM'*, *touch_method='MINITOUCH'*, *ime_method='YOSEMITEIME'*)
Bases: *airtest.core.device.Device*

Android Device Class

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

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

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*)
Input text on the device

Parameters

- **text** – text to input
- **enter** – True or False whether to press *Enter* key

Returns None

touch(*pos*, *times=1*, *duration=0.01*)
Perform touch event on the device

Parameters

- **pos** – coordinates (x, y)
- **times** – how many touches to be performed
- **duration** – how long to touch the screen

Returns None

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

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 all devices

Returns True or False

unlock()
Unlock the device

Notes

Might not work on all devices

Returns None

display_info
Return the display info (orientation, rotation and max values for x and y coordinates)

Returns display information

get_display_info()
Return the display physical info (orientation, rotation and max values for x and y coordinates)

Returns display physical information

get_current_resolution()
Return current resolution after rotaion

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. Recoding file will be kept in the device.

Parameters

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

Returns None

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'

ADBTOUCH = 'ADBTOUCH'

class IME_METHOD
Bases: object

ADBIME = 'ADBIME'

YOSEMITEIME = 'YOSEMITEIME'

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

text(*value*)
Input text with Yosemite input method

Parameters **value** – text to be inputted

Returns output form *adb shell* command

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

airtest.core.android.minicap module

class Minicap(*adb*, *projection=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(*inst*, **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(*inst*, **args*, ***kwargs*)
Get display info by minicap

> **Warning:** It might segfault, the preferred way is to get the information from adb commands

Returns display information

get_frame(*inst*, **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(*inst*, **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()

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

airtest.core.android.minitouch module

class Minitouch(*adb*, *backend=False*)

Bases: `object`

Super fast operation from minitouch

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

install_and_setup(*inst*, **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(*inst*, **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(*inst*, **args*, ***kwargs*)
Perform swipe event

minitouch protocol example:

```
d 0 0 0 50
c
m 0 20 0 50
c
m 0 40 0 50
c
m 0 60 0 50
c
m 0 80 0 50
c
m 0 100 0 50
c
u 0
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(*inst*, **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(*inst*, **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(*inst*, **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*)

airtest.core.android.recorder module

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

Screen recorder

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

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

airtest.core.android.rotation module

class RotationWatcher(*adb*)
Bases: `object`

RotationWatcher class

get_ready(*inst*, **args*, ***kwargs*)

teardown()

start()
Start the RotationWatcher daemon thread

Returns None

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)

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(*inst*, **args*, ***kwargs*)

uninstall()
Uninstall *Yosemite.apk* application from the device

Returns None

1.4 airtest.core.ios package

1.4.1 Submodules

airtest.core.ios.ios module

airtest.core.ios.minicap module

1.5 airtest.core.win package

This package provide Windows Client Class.

1.5.1 Submodules

airtest.core.win.screen module

screenshot(*filename*, *hwnd=None*)
Take the screenshot of Windows app

Parameters

- **filename** – file name where to store the screenshot
- **hwnd** –

Returns bitmap screenshot file

airtest.core.win.win module

require_app (*func*)

class Windows (*handle=None*, *dpifactor=1*, ***kwargs*)
Bases: *airtest.core.device.Device*

Windows client.

connect (*handle=None*, ***kwargs*)
Connect to window and set it foreground

Parameters ****kwargs** – optional arguments

Returns None

shell (*cmd*)
Run shell command in subprocess

Parameters **cmd** – command to be run

Raises `subprocess.CalledProcessError` – when command returns non-zero exit status

Returns command output as a byte string

snapshot (*filename='tmp.png'*)
Take a screenshot and save it to *tmp.png* filename by default

Parameters **filename** – name of file where to store the screenshot

Returns display the screenshot

keyevent (*keyname*, ***kwargs*)
Perform a key event

References

https://pywinauto.readthedocs.io/en/latest/code/pywinauto.keyboard.html

Parameters

- **keyname** – key event
- ****kwargs** – optional arguments

Returns None

text (*text*, ***kwargs*)
Input text

Parameters

- **text** – text to input
- ****kwargs** – optional arguments

Returns None

touch (*pos*, ***kwargs*)
Perform mouse click action

References

https://pywinauto.readthedocs.io/en/latest/code/pywinauto.mouse.html

Parameters

- **pos** – coordinates where to click
- ****kwargs** – optional arguments

Returns None

swipe(*p1*, *p2*, *duration=0.8*, *steps=5*)
Perform swipe (mouse press and mouse release) :param p1: start point :param p2: end point :param duration: time interval to perform the swipe action :param steps: size of the swipe step

Returns None

start_app(*path*)
Start the application

Parameters **path** – full path to the application

Returns None

stop_app(*pid*)
Stop the application

Parameters **pid** – proccess ID of the application to be stopped

Returns None

set_foreground(*inst*, **args*, ***kwargs*)
Bring the window foreground

Returns None

get_rect(*inst*, **args*, ***kwargs*)
Get rectangle

Returns None

get_title(*inst*, **args*, ***kwargs*)
Get the window title

Returns window title

get_pos(*inst*, **args*, ***kwargs*)
Get the window position coordinates

Returns coordinates of topleft corner of the window (left, top)

move(*inst*, **args*, ***kwargs*)
Move window to given coordinates

Parameters **pos** – coordinates (x, y) where to move the window

Returns None

kill(*inst*, **args*, ***kwargs*)
Kill the application

Returns None

focus_rect

get_current_resolution()

1.6 airtest

1.6.1 airtest package

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_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 __author__, __title__ and __desc__.

get_author_title_desc(*text*)
Get author title desc.

strip_str(*string*)
Strip string.

airtest.cli.parser module

get_parser()

runner_parser(*ap=None*)

airtest.cli.runner module

class AirtestCase(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

PROJECT_ROOT = '.'

SCRIPTEXT = '.air'

TPLEXT = '.png'

classmethod setUpClass()

setUp()

tearDown()

runTest()

classmethod exec_other_script(*scriptpath*)
run other script in test script

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(*screen=None*)
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.

RADIUS_X = 250

RADIUS_Y = 250

static count_record_pos(*pos*, *resolution*)

classmethod get_predict_point(*record_pos*, *screen_resolution*)

classmethod get_predict_area(*record_pos*, *screen_resolution*)

airtest.core.device module

class MetaDevice

Bases: type

REGISTRY = {'Device': <class 'airtest.core.device.Device'>, 'Android': <class 'airte

class Device

Bases: object

base class for test device

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

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

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

swipe(*t1*, *t2*, ***kwargs*)

keyevent(*key*, ***kwargs*)

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

start_app(*package*)

stop_app(*package*)

clear_app(*package*)

list_app(***kwargs*)

install_app(*uri*, ***kwargs*)

uninstall_app(*package*)

get_current_resolution()

airtest.core.error module

error classes

exception BaseError(*value*)

Bases: exceptions.Exception

exception AirtestError(*value*)

Bases: *airtest.core.error.BaseError*

This is Airtest BaseError

exception TargetNotFoundError(*value*)

Bases: *airtest.core.error.AirtestError*

This is TargetNotFoundError BaseError When something is not found

exception ScriptParamError(*value*)
Bases: *airtest.core.error.AirtestError*

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 \\'\\w+\\' 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.

And create dir at *dirpath/ST.SCREEN_DIR* for screenshots

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

Returns:

log(*tag*, *data*, *in_stack=True*)

log_in_func(*data*)

logwrap(*f*)

device_platform()

using(*path*)

import_device_cls(*platform*)
lazy import device class

on_platform(*platforms*)

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 = 0.6

THRESHOLD_STRICT = 0.7

OPDELAY = 0.1

FIND_TIMEOUT = 20

FIND_TIMEOUT_TMP = 3

airtest.report package

Submodules

airtest.report.report module

class LogToHtml(*script_root*, *log_root=''*, *static_root=''*, *author=''*, *export_dir=None*, *logfile='log.txt'*, *lang='en'*)

Bases: `object`

Convert log to html display

scale = 0.5

analyse()

translate(*step*)

depth = 1 name touch,swipe,wait,exist 3

3=screenshot 2= _loop_find 1=

keyevent,text,sleep 1=

assert exist

static to_percent(*p*)

static div_rect(*r*, *offset=None*)

static func_desc_zh(*step*)

(depth=1)name

static func_desc(*step*)

static func_title(*step*)

static dis_vector(*v*)

static dis_vector_zh(*v*)

render(*template_name*, *output_file=None*, *record_list=None*)

get_script_name(*path*)

get_file_author(*file_path*)

simple_report(*logpath*, *tplpath='.'*, *logfile='log.txt'*, *output='log.html'*)

get_parger(*ap*)

main(*args*)

Python Module Index

a

Index

D

E

F

G

R

S

T

U

V

W

X

Y