ChetBot Automation API Reference

Version 0.6.0

The ChetBot API provides a simple JavaScript interface to interact with your Android app. It makes use of lazily-evaulated view selectors to allow you to efficiently navigate your UI.

The API uses Rhino under the hood so you can access any Android or Java libraries available in your app.

View selectors

"Selectors" can be a view, list of views, a string which corresponds to the text shown by a view or an JavaScript object that matches a number of views.

An object selector can specify one or more attributes:

- `text` - The text displayed on the `TextView`.
- `id` - The Android ID of the view. Can be specified in its short form (e.g. `"password"`) or its fully namespaced form. (e.g. `com.domain.myapp:id/password`)
- `type` - The class

Some examples of selectors passed to `tap`:

- `tap('username')`
- `tap({text: 'username'})` - Equilvalent to above
- `tap(view('username'))` - Equilvalent to above
- `tap(view({text: 'username'}))` - Equilvalent to above
- `tap(topmost('username'))`
- `tap({id: 'password', type: 'EditText', text: ''})` - Finds an empty text field with an id matching `*:id/password`.
- `tap({id: 'com.domain.myapp:id/password', type: 'EditText', text: ''})` - Finds an empty text field with an id matching `com.domain.myapp:id/password`.

`topmost(view_selector)`

Find the view that's the closest to the top, matching `view_selector`.

`bottommost(view_selector)`

Find the view that's the closest to the bottom, matching `view_selector`.

`leftmost(view_selector)`

Find the view that's the closest to the left, matching `view_selector`.

`rightmost(view_selector)`

Find the view that's the closest to the right, matching `view_selector`.

`centermost(view_selector)`

Find the view that's the closest to the center, matching `view_selector`.

`outermost(view_selector)`

Find the view that's the furthest from the center, matching `view_selector`.

`location(view_selector)`

Returns a `[x, y]` pair of the location of the first selected view on the screen.

`view(view_selector)`

Select the first view matching `view_selector`.

`view(view_selector)`

Select all views matching `view_selector`.

Interaction

`home()`

Simulate a press of the device's "Home" key

`press(key)`

Press a key or button on the device. `key` can be:

- `'enter'`
- `'back'`
- `'backspace'`

`tap(view_selector)`

Tap the center of the selected view.

`open_drawer(side)`

Opens the slide-out drawer on the side given. `side` defaults to `'start'`.

`side` may be one of:

- `'start'` (The left side in a left-to-right environment)
- `'end'`

`close_drawer(side)`

Closes the slide-out drawer on the side given. `side` defaults to `'start'`.

`side` may be one of:

- `'start'` (The left side in a left-to-right environment)
- `'end'`

`exists(view_selector)`

Returns `true` if `view_selector` represents any views, `false` otherwise.

`count(view_selector)`

Returns the number of views represented by `view_selector`.

`text(view_selector)`

Returns the text displayed on the first selected view.

`class_of(view_selector)`

Returns the class of the first selected view.

`id(view_selector)`

Returns the fully namespace ID of the first selected view. (e.g. `'com.domain.myapp:id/password'`)

`size(view_selector)`

Returns a `[width, height]` array of the size of first the selected view in pixels.

wait

`wait(duration)`

Wait for the given number of seconds.

`wait_for(view_selector, options = {timeout: 60})`

Wait until `exists(view_selector)` is `true`. Throws an error after `options.timeout` seconds.

`wait_until_idle()`

Wait until the the view hierarchy is completely idle. Throws an error if not idle after ten seconds.

Assertions

`assert_true(value)`

Throw an error if `value` is not truthy.

`assert_false(value)`

Throw an error if `value` is truthy.

`assert_equal(a, b)`

Throw an error if `a` is not equivalent (`!=`) to `b`.

`assert_exists(view_selector)`

Equivalent to `assert_true(exists(view_selector))`.

Keyboard

`hide_keyboard()`

Dismiss the soft keyboard if it is showing

`type_text(text)`

Simulate typing the characters of `text` on a connected keyboard.

Utilities

`toast(text)`

Show a "Toast" on the screen for a few seconds, showing the text given.

`screenshot()`

Take a screenshot and show it in the test report.