boop.js Automation API Reference

Version 0.7.7

The boop.js API provides a simple JavaScript interface to interact with your Android app. It uses simple 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

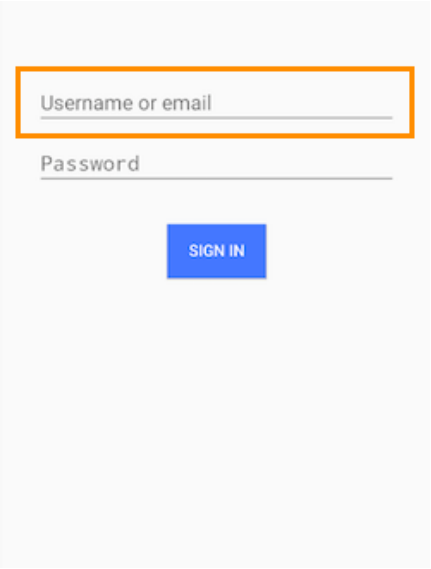
"Selectors" can be a view, list of views, a string which corresponds to the text shown by a view or an JavaScript object that matches a number of views.

An object selector can specify one or more attributes:

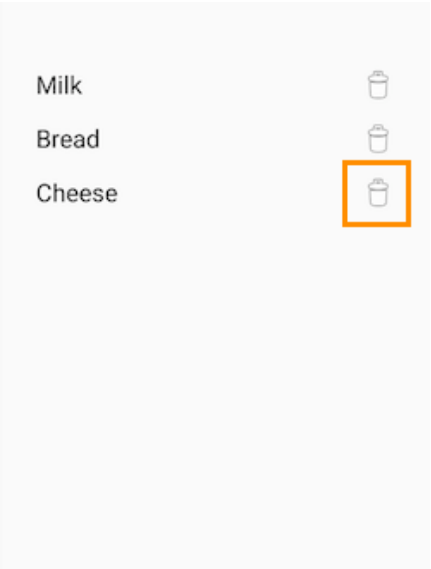
- `text` - The text displayed on the `TextView`. (Case insensitive)
- `has_text` - Partially match text in the `TextView`. (Case insensitive.)
- `id` - The Android ID of the view. Can be specified in its short form (e.g. `"password"`) or its fully namespaced form. (e.g. `com.domain.myapp:id/password`)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

Common view selectors patterns

Any of the following view selectors match view highlighted.




```
'username or email'

{text: 'username or email'}

{has_text: 'email'}

topmost({type: 'EditText'})

/^user/i

function(v) {
  return text(v) === 'Username or email';
}
```




```
views({id: 'delete'}).closest_to('cheese')

views({type: 'ImageButton'})[2]

bottommost({id: 'delete'})

bottommost({type: 'ImageButton'})
```

Some examples of selectors passed to `tap`:

- `tap('username')` - Tap view with the text "username"
- `tap({text: 'username'})` - Equilvalent to above
- `tap(view('username'))` - Equilvalent to above
- `tap(view({text: 'username'}))` - Equilvalent to above
- `tap({text: /^user/i})` - Tap a view whose text matches the regular expression
- `tap({has_text: 'Login now'})` - Tap a view that contains the phrase "Login now"
- `tap(topmost('username'))` - Tap the topmost view with the text "username"
- `tap({id: 'password', type: 'EditText', text: ''})` - Tap an empty text field with an id matching `.*:id/password`.
- `tap({id: 'com.domain.myapp:id/password', type: 'EditText', text: ''})` - Tap 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.

`[ ... ].closest_to(label)`

Find the view closest to the first view in the view selector given. e.g.

```
views('delete').closest_to('bananas')
```

`[ ... ].furthest_from(label)`

Find the view furthest frem the first view in the view selector given.

`view(view_selector)`

Select the first view matching `view_selector`.

`views(view_selector)`

Select all views matching `view_selector`.

Activity

`activity()`

Get the current activity.

`content_view()`

Get the topmost content view visible on the application. This is usually the application view itself, but may be a popup window if one is visible.

`run_on_ui_thread(fn)`

Runs the function `fn` on the UI thread.

`open_uri(uri)`

Opens the URI `uri` using the default system URI handler.

Note that this may take you away from the application. You will be unable to control another application using the API described in this document.

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.

`tap([x, y])`

Tap the `x` `y` location on the screen.

`swipe_left([view_selector])`

Swipe left starting at the view provided, otherwise start at the center of the screen.

`swipe_right([view_selector])`

Swipe right starting at the view provided, otherwise start at the center of the screen.

`swipe_up([view_selector])`

Swipe up starting at the view provided, otherwise start at the center of the screen.

`swipe_left([view_selector])`

Swipe down starting at the view provided, otherwise start at the center of the screen.

`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'`

`visible(view_selector)`

Returns `true` if `view_selector` represents any visible 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.

`type(view_selector)`

Returns the class of the first selected view.

`id(view_selector)`

Returns the fully namespaced 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(wait_for_fn, options = {timeout: 60})`

Wait until `wait_for_fn()` is `truthy`. If `wait_for_fn` is not a function, assumes it is a selector passed to `view()`. Throws an error after `options.timeout` 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_visible(view_selector)`

Equivalent to `assert_true(visible(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.

`in_webview(selector, fn, callback)`

Run `fn` in the WebView selected by `selector`.

The optional `callback` is called with the result of `fn`.

e.g.

```
in_webview({type: 'WebView'}, function() {
  $('a.login').click();
});
```