SODA JSON SYNTAX

Soda syntax overview and JSON key/value cheat-sheet for frameworks using the *core/mobile v2.0* syntax.

OVERVIEW

Soda is a flexible testing framework with support for custom syntaxes. Each Soda syntax can structure its JSON layout in any way it desires—that is, with one exception: each syntax must include a *meta* key at its root with the following structure:

```
{
    meta: {
        "name": "file name",
        "description": "file description",
        "syntax": {
            "name": " mobile ",
            "version": "syntax version"
        }
    }
    ...
}
```

MOBILE v2.0 SYNTAX

The JSON layout for the Mobile v2.0 Syntax. Each respective file type should have the following structure…

Actions

```
{
    “meta”: {
        "name": "file name",
        "description": "file description",
        “id” : “optional id”
        "syntax": {
            "name": "mobile",
            "version": "2.0"
        }
    }
    “actions”: [
        {
            “key”: value,
            “key”: value,
            ...
        },
        {
            “key”: value,
            “key”: value,
            ...
        },
        ...
    ]
}
```

Screens

```
{
    “meta”: {
        "name": "file name",
        "description": "file description",
        “id” : “optional id”
        "syntax": {
            "name": "mobile",
            "version": "2.0"
        }
    }
    “screen”: {
        “components”: [
            {
                “key”: value,
                “key”: value,
                ...
            },
            {
                “key”: value,
                “key”: value,
                ...
            },
            ...
        ]
    }
}
```

Menus

```
{
    “meta”: {
        "name": "file name",
        "description": "file description",
        “id” : “optional id”
        "syntax": {
            "name": "mobile",
            "version": "2.0"
        }
    }
    “menu”: {
        “components”: [
            {
                “key”: value,
                “key”: value,
                ...
            },
            {
                “key”: value,
                “key”: value,
                ...
            },
            ...
        ]
    }
}
```

Popups

```
{
    “meta”: {
        "name": "file name",
        "description": "file description",
        “id” : “optional id”
        "syntax": {
            "name": "mobile",
            "version": "2.0"
        }
    }
    “popup”: {
        “components”: [
            {
                “key”: value,
                “key”: value,
                ...
            },
            {
                “key”: value,
                “key”: value,
                ...
            },
            ...
        ]
    }
}
```

USING VARIABLES

Within strings in any action file, you can use variables to replace or inject values. Variables are specified by the *${variable_name}* syntax, where *variable_name* is the name of the variable you wish to use. If the variable hasn’t been set and is referenced, *null* is returned.

A variable reference must be wrapped in curly braces.
You can set variables by using the “*store as*” and “*save as*” action objects as described below.

QUICK REFERENCE

A list of the available Mobile v2.0 actions and screen assertions.
See the detailed tables on the following pages for the available key sets for each listing and examples.

ACTIONS		SCREENS/MENUS/POPUPS	
swipe	tap	assert exists	assert is
hideAppFor	typeOnKeyboard	assert matches	assert hasCount
back	debug		
refresh	execute		
validate	waitFor		
wait	store as		
save as	set to		
scroll	scrollToVsible		
retries	setConfig		

CORE SYNTAX: SCREEN/MENU/POPUP VALIDATIONS

Assertions to validate screens, menus and popups that can be used by any syntax. Mobile v2.0 includes all of these.

JSON PATH(S)	KEYS/TYPES	DESCRIPTION
assert exists		

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	exists *boolean*
	using *string*
		Asserts that an element exist or doesn’t exist using *using* (*selector*, *name*, *id*, *value*) to find the element. Key *exists* can be either *true* or *false*.
		Example
		`{`
		`    assert: “element name”,`
		`    exists: true,`
		`    using: “name”`
		`}`

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	exists *boolean*
		Asserts that an element exist or doesn’t exist using a selector.
		Example
		`{`
		`    assert: “selector”,`
		`    exists: true`
		`}`

assert matches		

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	matches *regular expression*
	property *string*
		Asserts that the specified *property* of an element matches the given regular expression.
		Example
		`{`
		`    assert: “selector”,`
		`    matches: “\\d+”,`
		`    property: “name”`
		`}`

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	matches *regular expression*
	property *string*
	using *string*
		Asserts that the specified *property* of an element matches the given regular expression using the value of *using* to find the element.
		Example
		`{`
		`    assert: “element id”,`
		`    matches: “\\d+”,`
		`    property: “value”,`
		`    using: “id”`
		`}`

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	matches *regular expression*
	using *string*
		Asserts that the *value* of an element matches the given regular expression using the value of *using* to find the element.
		Example
		`{`
		`    assert: “element id”,`
		`    matches: “\\d+”,`
		`    using: “id”`
		`}`

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	matches *regular expression*
		Asserts that the *value* of an element with the given *selector* matches the given regular expression.
		Example
		`{`
		`    assert: “element id”,`
		`    matches: “\\d+”,`

		}
assert is		

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	is *string*
	property *string*
		Asserts that the specified *property* of an element is equal to the value of *is.*
		Example
		{
		    assert: "selector",
		    is: "\\d+",
		    property: "name"
		}

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	is *string*
	property *string*
	using *string*
		Asserts that the specified *property* of an element is equal to the value of *is* using the value of *using* to find the element.
		Example
		{
		    assert: "element id",
		    is: "\\d+",
		    property: "value",
		    using: "id"
		}

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	is *string*
	using *string*
		Asserts that the *value* of an element is equal to the value of *is* using the value of *using* to find the element.
		Example
		{
		    assert: "element id",
		    is: "\\d+",
		    using: "id"
		}

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	is *string*
		Asserts that the *value* of an element is equal to the value of *is.*
		Example
		{
		    assert: "element selector",
		    is: "\\d+",
		}

assert hasCount		

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	hasCount *string|number*
	using *string*
		Asserts that the returned set of elements contains *hasCount* elements, using the value of *using* to find the elements.
		Example
		{
		    assert: "elementsWithName",
		    hasCount: 5,
		    using: "name"
		}

screen/components/*/
menu/components/*/
popup/components/*/
	assert *string*
	hasCount *string|number*
		Asserts that the returned set of elements contains *hasCount* elements, using the specified selector to find the elements.
		Example
		{
		    assert: "elementsWithSelector",
		    hasCount: 5,
		}

CORE SYNTAX: ACTIONS

Actions that can be used by any syntax. Mobile v2.0 includes all of these.

JSON PATH(S)	KEYS/TYPES	DESCRIPTION
debug		
actions/*/	debug *	Prints the *value* to the screen, useful for debugging **Example** `{ debug: “hello world!” }` **Example** `{ debug: “${variable}” }`
refresh		
actions/*/	refresh *boolean*	Refreshes the DOM Tree. *Use this only if all else fails—frequent refreshing will slow tests!* **Example** `{ refresh: true }`
retries		
actions/*/	retries *integer*	Sets the number of element retires from that point in the test forward. **Example** `{ retries: 25 }`
execute		
actions/*/	execute *string* module *string* type *string*	Executes the provided action from the specified *module* of type *type.* Type should only contain one of the values: *action* or *test.* **Example** `{ execute: “action filename”, module: “my module”, type: “action” }`
actions/*/	execute *string* type *string*	Executes the provided action or test from the current module of type *type.* Type should only contain one of the values: *action* or *test.* **Example** `{ execute: “action filename”, type: “test” }`
actions/*/	execute *string* repeat *integer* type *string*	Executes the provided action or test from the current module of type *type, repeat* times. If repeat is greater than 1, the action file or test file will be executed multiple times. **Example** `{`

		`execute: "action filename",`
		`repeat: 7,`
		`type: "action"`
		`}`

actions/*/	execute *string*
	repeat *integer*
		Executes the provided action from the current module *repeat* times. If repeat is greater than 1, the action file or test file will be executed multiple times.
		Example
		`{`
		`execute: "action filename",`
		`repeat: 7`
		`}`

actions/*/	execute *string*
	module *string*
		Executes the provided action from the specified *module.*
		Example
		`{`
		`execute: "action filename",`
		`module: "my module"`
		`}`

actions/*/	execute *string*	Executes the provided action from the current module
		Example
		`{`
		`execute: "action filename"`
		`}`

validate		

actions/*/	validate *string*
	type *string*
		Validates the specified screen/menu/popup of type *type.*
		Example
		`{`
		`validate: "my popup",`
		`type: "popup"`
		`}`
		Example
		`{`
		`validate: "my screen",`
		`type: "screen"`
		`}`

actions/*/	validate *string*	Validates the specified **screen**.
		Example
		`{`
		`validate: "my screen"`
		`}`

waitFor		

actions/*/	waitFor *string*
	using *string*
		Waits for the specified element(s), found by the value of *using*, before continuing to the next test item. Throws a failure if the element isn't found in *config.findElementRetries* times
		Example
		`{`
		`waitFor: "elementsWithName",`
		`using: "name"`
		`}`

actions/*/	waitFor *string*	Waits for the specified element(s), found by selector, before continuing to the next test item. Throws a failure if the element isn't found in *config.findElementRetries* times **Example**

		`{`
		`    waitFor: “elementsWithSelector”`
		`}`

wait		

actions/*/	wait *integer*	Waits *wait* seconds before continuing to the next test item.
		Example
		`{`
		`    wait: 20`
		`}`

store as		

actions/*/	store *string|number|object*
	as *string*
		Stores the value of *store* as a variable named *as*
		Example
		`{`
		`    store: “hello world!”,`
		`    as: “foo”`
		`}`

actions/*/	store *string|number|object*
	as *string*
	capture *regular expression*
		Stores the value of *store* as a variable named *as*, capturing values in an array using the provided regular expression.
		
		*If the capture doesn’t return any non-null captures, the test will throw a failure.
		
		Example
		`{`
		`    store: “hello world!”,`
		`    as: “foo”,`
		`    capture: “(hello).*!”`
		`}`

actions/*/	store *string|number|object*
	as *string*
	capture *regular expression*
	index *integer*
		Stores the value of *store* as a variable named *as*, capturing values in an array using the provided regular expression, and saving only the index specified from the capture array.
		
		*If the index doesn’t exist, the test will throw a failure.
		Example
		`{`
		`    store: “hello world!”,`
		`    as: “foo”,`
		`    capture: “(hello).*!”,`
		`    index: 0`
		`}`

set to *These will throw a failure if more than one element is returned in the found set.		

actions/*/	set *string*
	to *string|number*
		Set the value of the element with the selector *set* to the value *to*
		Example
		`{`
		`    set: “elementWithSelector”,`
		`    to: “some value”`
		`}`

actions/*/	set *string*
	to *string|number*
	using *string*
		Set the value of the element found with *using* to the value *to*
		Example
		`{`
		`    set: “elementWithName”,`
		`    using: “name”,`
		`    to: “some value”`
		`}`

scrollToVisible
*These will throw a failure if more than one element is returned in the found set. **Valid for iOS only**

actions/*/	scrollToVisible
	using
		string
		string|number
			Scroll the element found by *using* into view.
			Example
			`{`
			`scrollToVisible: “elementWithSelector”,`
			`using: “selector”`
			`}`

actions/*/	scrollToVisible	*string*	Scroll the element with the specified selector into view.
			Example
			`{`
			`scrollToVisible: “elementWithSelector”`
			`}`

scroll
*These will throw a failure if more than one element is returned in the found set. **Not valid for iOS.**

actions/*/	scroll
	amount
	direction
	refresh
	using
		string
		number
		string
		boolean
		string
			Scroll an element *amount* in direction *direction*, found by the value of *using*.
			
			If refresh is true, the DOM Tree will be updated after the scroll.
			
			Amount is based on the element height: 1 is equivalent to scrolling the element it’s entire height. So if the element is 50px tall, 1 would scroll in *direction* 50px.
			
			Direction can be *up*, *down*, *left* or *right*.
			Example
			`{`
			`scroll: “elementWithId”,`
			`amount: 5,`
			`direction: “down”,`
			`refresh: false,`
			`using: “id”`
			`}`

actions/*/	scroll
	amount
	direction
	using
		string
		string|number
		string
		string
			Scroll an element *amount* in direction *direction*, found by the value of *using*.
			
			Amount is based on the element height: 1 is equivalent to scrolling the element it’s entire height. So if the element is 50px tall, 1 would scroll in *direction* 50px.
			
			Direction can be *up*, *down*, *left* or *right*.
			Example
			`{`
			`scroll: “elementWithId”,`
			`amount: 5,`
			`direction: “down”,`
			`using: “id”`
			`}`

actions/*/	scroll
	direction
	using
		string
		string
		string
			Scroll an element in direction *direction*, found by the value of *using*.
			
			Amount will default to 1.
			The DOM Tree will automatically be refreshed.
			
			Direction can be *up*, *down*, *left* or *right*.
			Example
			`{`
			`scroll: “elementWithId”,`

		`direction: “down”,`
		`using: “id”`
		`}`

actions/*/	scroll *string*
	direction *string*
	amount *string|number*
		Scroll an element with the specified selector *amount* in direction *direction*.
		
		Amount will default to 1.
		The DOM Tree will automatically be refreshed.
		
		Direction can be *up*, *down*, *left* or *right*.
		Example
		`{`
		`scroll: “elementWithSelector”,`
		`amount: 5,`
		`direction: “down”`
		`}`

actions/*/	scroll *string*
	direction *string*
		Scroll an element with the specified selector in direction *direction*.
		
		Amount will default to 1.
		The DOM Tree will automatically be refreshed.
		
		Direction can be *up*, *down*, *left* or *right*.
		Example
		`{`
		`scroll: “elementWithSelector”,`
		`direction: “left”`
		`}`

actions/*/	scroll *string*
	amount *string|number*
		Scroll an element with the specified selector down *amount*.
		The DOM Tree will automatically be refreshed.
		Example
		`{`
		`scroll: “elementWithSelector”,`
		`amount: 5.478`
		`}`

actions/*/	scroll *string*	Scroll an element with the specified selector down.
		The DOM Tree will automatically be refreshed.
		Example
		`{`
		`scroll: “elementWithSelector”`
		`}`

save as *These will throw a failure if more than one element is returned in the found set.		

actions/*/	save *string*
	as *number*
	capture *string*
	index *integer*
	using *string*
	property *string*
		Save the property *property* of element found with *using* as a variable named *as*, capturing an array specified by the regular expression *capture*, and storing only index *index* from the capture array.
		Example
		`{`
		`save: “elementWithLabel”,`
		`as: “variableName”,`
		`capture: “(some)(regular)(expression)”,`
		`index: 5,`
		`using: “label”,`
		`property: “parent.id”`
		`}`

actions/*/	save *string*	Save the property *property* of element found with *using* as a

	as *number*
	using *string*
	property *string*
		variable named *as*.
		
		This could result in a stored string, number, object, or array.
		Example
		`{`
		`    save: “elementWithLabel”,`
		`    as: “variableName”,`
		`    using: “label”,`
		`    property: “parent”`
		`}`

actions/*/	save *string*
	as *number*
	capture *string*
	index *integer*
	property *string*
		Save the property *property* of element found with the specified selector as a variable named *as*, capturing an array specified by the regular expression *capture*, and storing only index *index* from the capture array.
		Example
		`{`
		`    save: “elementWithSelector”,`
		`    as: “variableName”,`
		`    capture: “(some)(regular)(expression)”,`
		`    index: 5,`
		`    property: “parent.id”`
		`}`

actions/*/	save *string*
	as *number*
	capture *string*
	using *string*
	property *string*
		Save the property *property* of element found with *using* as a variable named *as*, capturing an array specified by the regular expression *capture*.
		
		** This will store an array.
		Example
		`{`
		`    save: “elementWithLabel”,`
		`    as: “variableName”,`
		`    capture: “(some)(regular)(expression)”,`
		`    using: “label”,`
		`    property: “parent.id”`
		`}`

actions/*/	save *string*
	as *number*
	capture *string*
	property *string*
		Save the property *property* of element found with the specified selector as a variable named *as*, capturing an array specified by the regular expression *capture*.
		
		** This will store an array.
		Example
		`{`
		`    save: “elementWithSelector”,`
		`    as: “variableName”,`
		`    capture: “(some)(regular)(expression)”,`
		`    index: 5,`
		`    using: “label”,`
		`    property: “parent.id”`
		`}`

actions/*/	save *string*
	as *number*
	property *string*
		Save the property *property* of element found with the specified selector as a variable named *as*.
		Example
		`{`
		`    save: “elementWithSelector”,`
		`    as: “variableName”,`
		`    property: “rect.origin.x”`
		`}`

actions/*/	save *string* as *number*	Save the **entire element** found with *using* as a variable named *as*.

	using *string*	**Example**
		`{`
		`    save: “elementWithSelector”,`
		`    as: “variableName”,`
		`    using: “selector”`
		`}`

actions/*/	save *string*
	as *number*
		Save the **entire element** found with the specified selector as a variable named *as*.
		Example
		`{`
		`    save: “elementWithSelector”,`
		`    as: “variableName”`
		`}`

MOBILE v2.0 SPECIFIC SYNTAX

Mobile v2.0 syntax only. Elements and device interactions that can be performed by a user on an element or device.

JSON PATH(S)	KEYS/TYPES	DESCRIPTION
swipe *These will throw a failure if more than one element is returned in the found set.		

actions/*/	swipe *string*
	options *object*
	refresh *boolean*
	using *string*
		Swipe an element, found by using, with options. If *refresh* is true, the DOM tree will be updated after the swipe, if false it will not.
		Example
		`{`
		`    swipe: “element id”,`
		`    options: {`
		`        direction: “up”`
		`    }`
		`    refresh: false,`
		`    using: “id”`
		`}`

actions/*/	swipe *string*
	options *object*
	using *string*
		Swipe an element, found by the value of *using*, with options. The DOM will automatically be updated after the swipe.
		Example
		`{`
		`    swipe: “element id”,`
		`    options: {`
		`        direction: “up”`
		`    }`
		`    refresh: false,`
		`}`

actions/*/	swipe *string*
	options *object*
	refresh *boolean*
		Swipe an element, found by selector, with options. If *refresh* is true, the DOM tree will be updated after the swipe, if false it will not.
		Example
		`{`
		`    swipe: “element id”,`
		`    options: {`
		`        direction: “up”`
		`    }`
		`    refresh: false,`
		`}`

actions/*/	swipe *string*
	options *object*
		Swipe an element with selector *swipe*, with options. The DOM will automatically be updated after the swipe.
		Example
		`{`
		`    swipe: “elementsWithSelector”,`
		`    options: {`
		`        direction: “up”`
		`    }`
		`}`

actions/*/	swipe *string*
	using *string*
		Swipe an element with the default options using *using* to find the element. The DOM will automatically be updated after the swipe.
		Example
		`{`
		`    swipe: “elementWithLabel”,`
		`    using: “label”`
		`}`

actions/*/	swipe *string*
	refresh *boolean*
		Swipe an element with the given selector. If *refresh* is true, the DOM tree will be updated after the swipe, if false it will not.
		Example
		`{`
		`    swipe: “elementWithSelector”,`
		`    refresh: false`
		`}`

actions/*/	swipe *string*	Swipe an element with the given selector.
		Example
		`{`
		`    swipe: “elementWithSelector”`
		`}`

tap *These will throw a failure if more than one element is returned in the found set.		

actions/*/	tap *string*
	options *object*
	refresh *boolean*
	using *string*
		Tap an element, found by using, with options. If *refresh* is true, the DOM tree will be updated after the swipe, if false it will not.
		Example
		`{`
		`    tap: “element id”,`
		`    options: {`
		`        direction: “up”`
		`    }`
		`    refresh: false,`
		`    using: “id”`
		`}`

actions/*/	tap *string*
	options *object*
	using *string*
		Tap an element, found by the value of *using*, with options. The DOM will automatically be updated after the swipe.
		Example
		`{`
		`    tap: “element id”,`
		`    options: {`
		`        direction: “up”`
		`    }`
		`    refresh: false,`
		`}`

actions/*/	tap *string*
	options *object*
	refresh *boolean*
		Tap an element, found by selector, with options. If *refresh* is true, the DOM tree will be updated after the swipe, if false it will not.
		Example
		`{`
		`    tap: “element id”,`
		`    options: {`
		`        duration: 5`
		`    }`
		`    refresh: false,`
		`}`

actions/*/	tap *string*
	options *object*
		Tap an element with selector *swipe*, with options. The DOM will automatically be updated after the swipe.
		Example
		`{`
		`    tap: “elementsWithSelector”,`
		`    options: {`
		`        direction: “up”`
		`    }`
		`}`

actions/*/	tap *string* using *string*	Tap an element with the default options using *using* to find the element. The DOM will automatically be updated after the swipe.

		Example
		`{`
		`    tap: “elementWithLabel”,`
		`    using: “label”`
		`}`

actions/*/	tap *string*
	refresh *boolean*
		Tap an element with the given selector. If *refresh* is true, the DOM tree will be updated after the swipe, if false it will not.
		Example
		`{`
		`    tap: “elementWithSelector”,`
		`    refresh: false`
		`}`

actions/*/	tap *string*	Tap an element with the given selector.
		Example
		`{`
		`    tap: “elementWithSelector”`
		`}`

hideAppFor		

actions/*/	hideAppFor *integer*	Puts the app in the background for the specified number of seconds, and then returns it to the foreground.
		Example
		`{`
		`    hideAppFor: 5`
		`}`

typeOnKeyboard		

actions/*/	typeOnKeyboard *integer*	Types on the on-screen keyboard (soft keyboard) the specified value. *This is differs than set, as it will actually tap the keys, rather than just setting an element value.*
		Example
		`{`
		`    typeOnKeyboard: “hello world!”`
		`}`

back *Not available for the Instruments (iOS) framework.		

actions/*/	back *	Presses the physical back key on the device (not valid on iOS devices). The value of the back key doesn’t matter.
		Example
		`{`
		`    back: true`
		`}`