SENCHACOMMUNITYDAYS 19

Workshop - TDD with Siesta

Bryntum

@bryntum

Workshop objectives

- Setting up the Siesta application
- Using the Siesta documentation
- Writing a basic unit test
- TDD / BDD
- Functional UI testing
- Application tests
- Monkey tests
- Test automation, reports, CI
- IDE integration, Cloud testing

SENCHACOMMUNITYDAYS 19

Prerequisites

- Basic JavaScript knowledge
- WebServer installed/running
- Siesta downloaded and up and running
- Ext JS downloaded & unzipped

SENCHACOMMUNITYDAYS 19

About me

- Ext JS developer since 2007
- Started Bryntum 2009
- Gantt UI Components & tools for enterprise web apps
- www.bryntum.com
- @bryntum

Raise of hands

- **Ext JS experience**
  - 0-3 years
  - 3-5 years
  - 5+ years
  - Previous experience of JS testing? Jasmine, Selenium?

Why test?

Productivity & Confidence

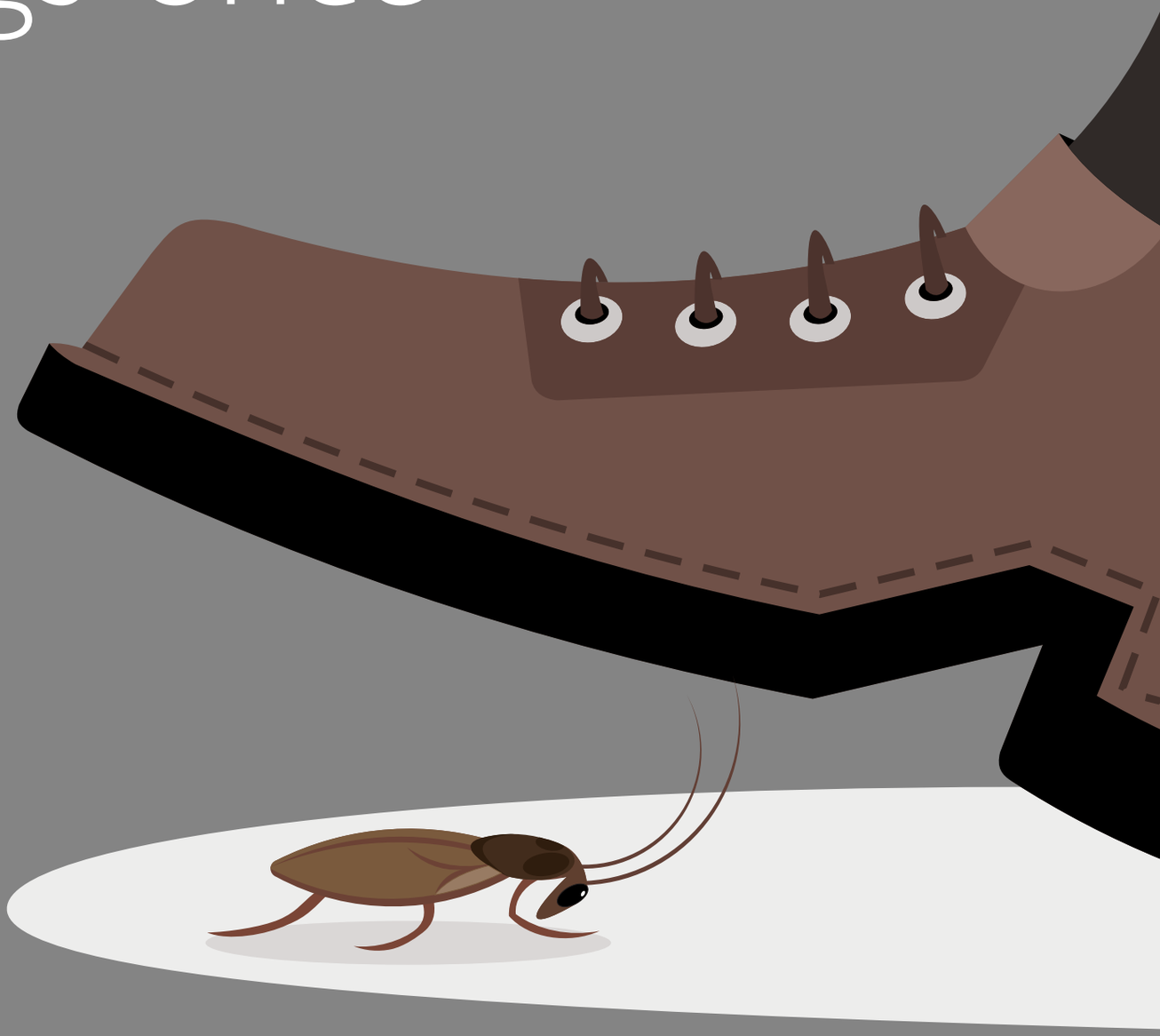
Productivity & Confidence

Code handover

Refactoring without tests

Refactoring like a **boss**

Fix bugs once

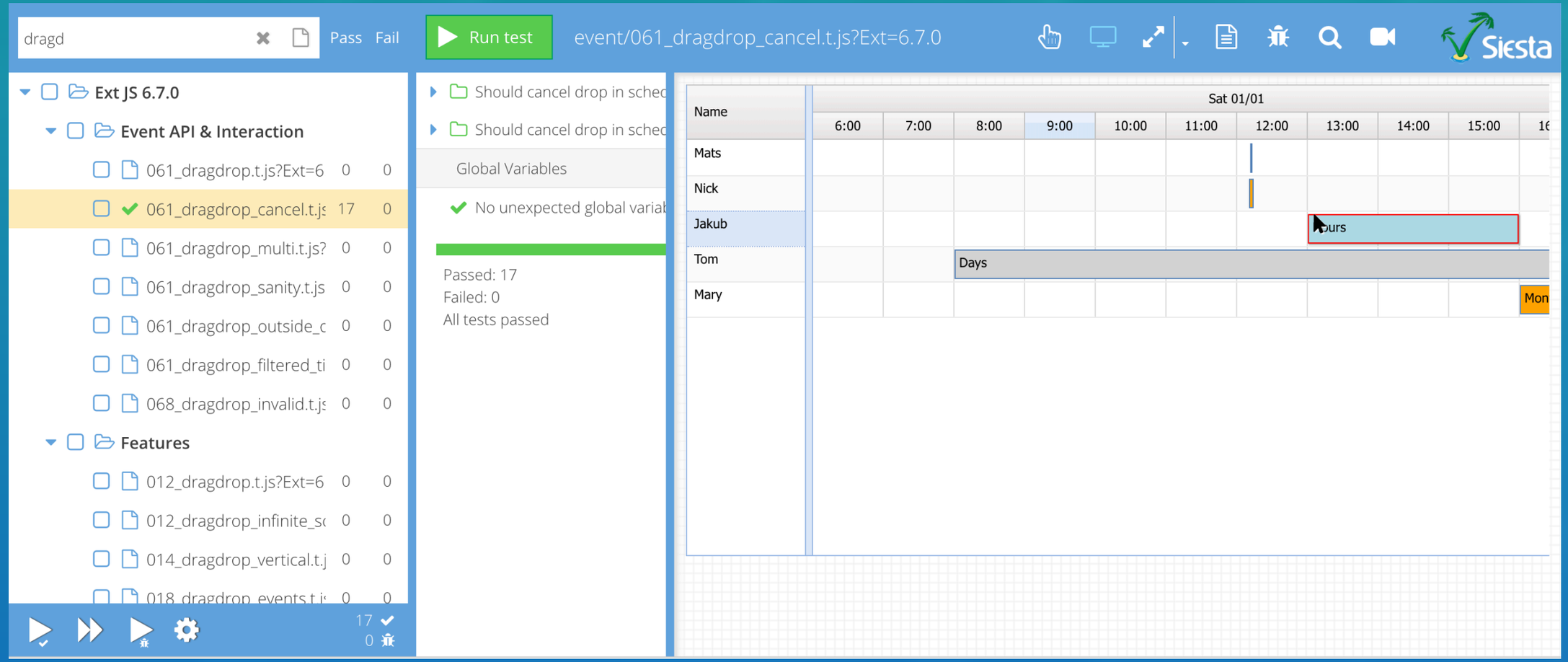


Quality of sleep

Setting up Siesta

Setting up Siesta

- Download Siesta ZIP archive
- Unzip, open /examples/browser in a running web server



Siesta docs

www.bryntum.com/docs/siesta/

Siesta API documentation

Search

Welcome to Siesta!

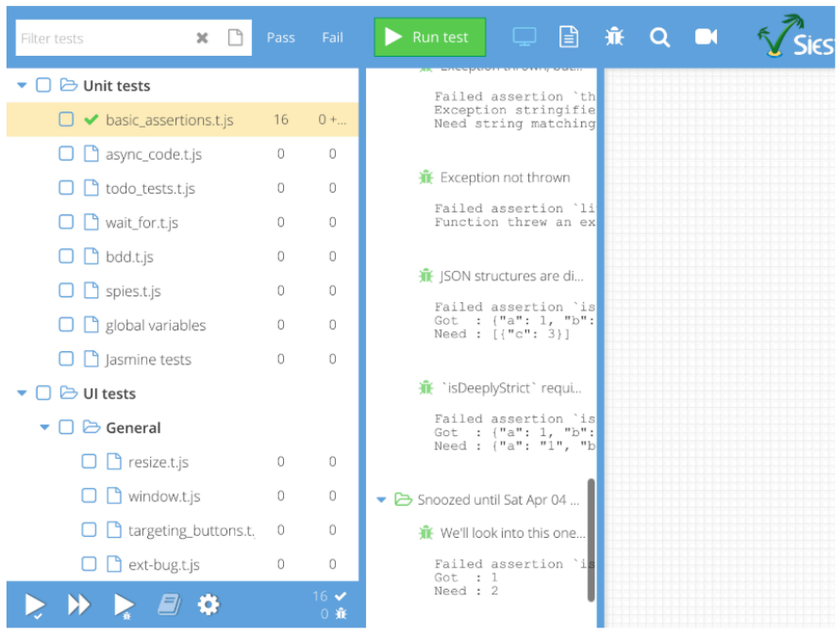
Siesta is a stress-free JavaScript unit and UI testing tool.

It is very easy to learn and as your test suite grows and your requirements becomes more complex Siesta still scales very well.

It will dramatically increase your confidence in your codebase and improve your quality of sleep, we promise.

Read the Getting Started Guide

Discuss on the forum



Learn more on bryntum.com

API Docs

Release Notes

Siesta overview

Siesta features

- Unit testing + Functional testing
- Each test is (optionally) sandboxed in its own iframe (Take a quick break and assert this using the DOM inspector)
- Written completely in JS, extensible.
- Test any webpage, JS or NodeJS code
- Adapter for Ext JS

Siesta features

- Automatically detects global variable leaks
- Support for async testing
- BDD layer (it/describe/expect/beforeEach etc)
- Code coverage (via Istanbul)
- Continuous integration support: TeamCity, Jenkins etc

SENCHACOMMUNITYDAYS 19

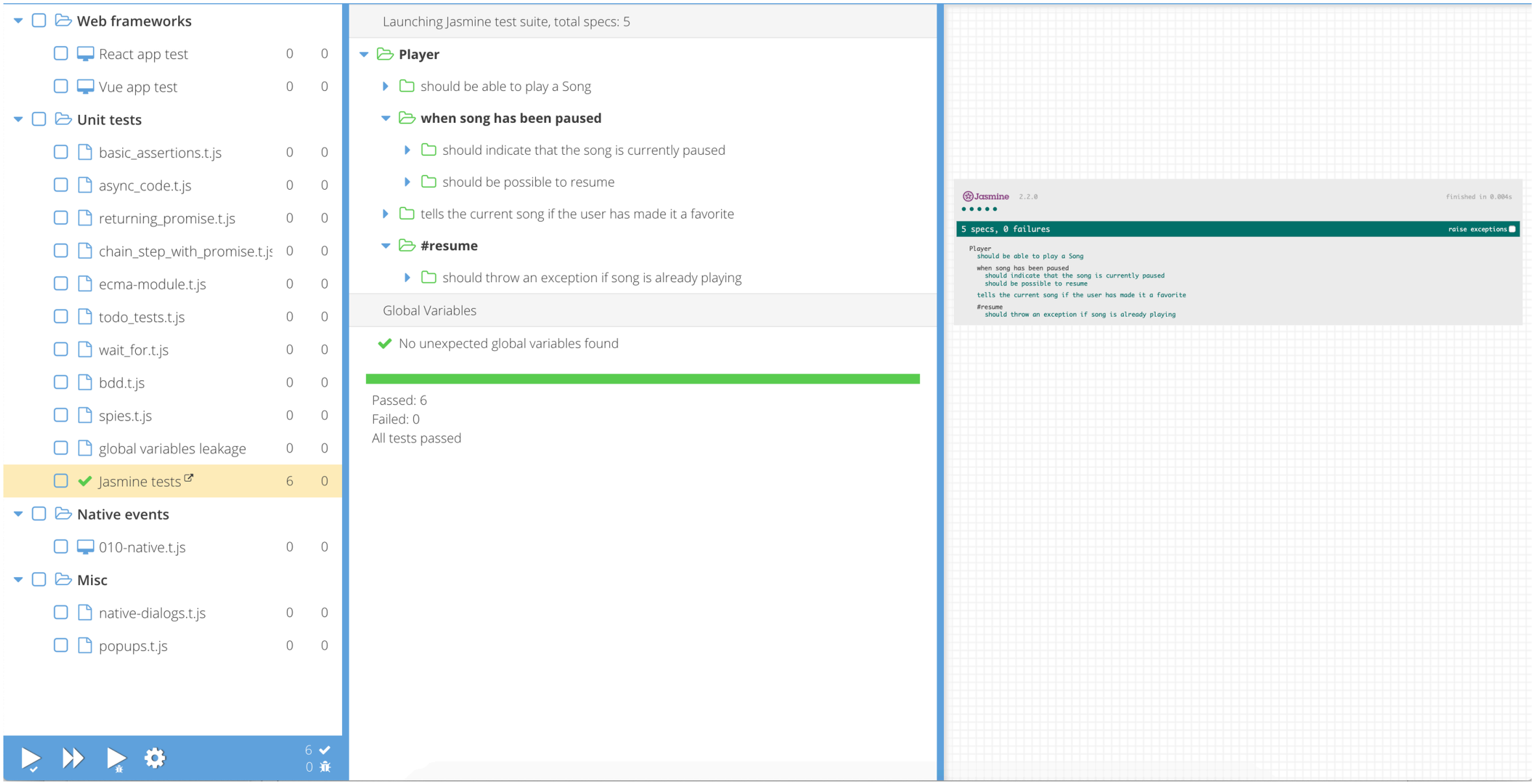
Sencha awareness

```
t.waitForStoresToLoad(userStore, callback);

t.waitForRowsVisible("usergrid", callback);

t.waitForEvent(someList, "selectionchange", cb);
```

Siesta UI

- Siesta core is built with Joose
- Siesta UI is built using Ext JS 6
- The upcoming Siesta 6 will feature a new UI + ease of use on mobile devices

Jasmine support



Jasmine support

```
{
    name                : 'Jasmine tests',
    jasmine             : true,
    expectedGlobals : ['Player', 'Song'],
    // url should point to the specs runner html
    url                 : '/jasmine_suite/SpecRunner.html'
}
```

Siesta Project

Siesta Project

- The Project is the Siesta application (with or without UI).
- The project UI shows a dashboard with the results of the test suite.
- `Siesta.Project.Browser` for plain HTML pages
- `Siesta.Project.Browser.ExtJS` for Ext apps
- https://www.bryntum.com/docs/siesta/#!/api/Siesta.Project.Browser
- `configure` & `start` methods

Project.configure

- Project.configure sets up global settings used for all tests
- Configs can be overridden on either test or group level.
- Tests can run in parallel or sequential (using "runCore" cfg)
- Set a **title** to persist UI state

```
var Project = new Siesta.Project.Browser.ExtJS();

Project.configure({
    autoCheckGlobals : true,
    expectedGlobals  : [
        'App'
    ],

    preload : [
        "ext-all.css",
        "ext-all-debug.js",
        { text : "window.ISTEST = true;" },
        "your-app-all.js"
    ]
});
```

Project configs

- transparentEx
- autoCheckGlobals
- expectedGlobals
- autoRun
- breakOnFail
- debuggerOnFail
- defaultTimeout
- disableCaching
- loaderPath / requires
- maxThreads
- speedRun
- viewportHeight/viewportWidth

Preloads

Loads the specified JS/CSS resources into the test

Same as including scripts/CSS in a HTML page

```
preload : [
        "ext-all.css",
        "ext-all-debug.js",
        { text : "window.ISTEST = true;" },
        "your-app-all-debug.js"
    ]
```

```
preload : [
      "ext-all.css",
      "ext-all-debug.js",
      { text : "window.ISTEST = true;" },
      "your-app-all.js"
 ]
```

Equals

```
<head>
    <link href="ext-all.css" rel="stylesheet" type="text/css"/>

    <script type="text/javascript" src="ext-all-debug.js"></script>
    <script type="text/javascript">
        window.ISTEST = true;
    </script>

    <script type="text/javascript" src="your-app-all.js"></script>
</head>
```

Project.start

- `Project.start` starts your project
- As input, provide your test suite (as a tree structure)
- Tests can be grouped in any way you like
- On each level in the tree you can override configs

Project.start

```
Project.start(
    {
        group          : 'Sanity tests',
        autoCheckGlobals : false,
        items          : [
            'sanity/010_range.t.js',
            'sanity/020_event.t.js'
        ]
    },
    ....
);
```

SENCHACOMMUNITYDAYS 19

Questions?

Siesta Tests

Two distinct types of tests

1. **Unit test:**
   - Focus on a particular class
   - Preload the files you want to test and focus only on those.
2. **Application test:**
   - Tell Siesta to go to your application URL
   - Loads all your files, CSS, JS and starts application
   - Interact with the page.

A Siesta test

- Is pure JS, can contain any code
- Should be wrapped by either `StartTest` (or `describe`)
- The `StartTest` method is called when the test starts

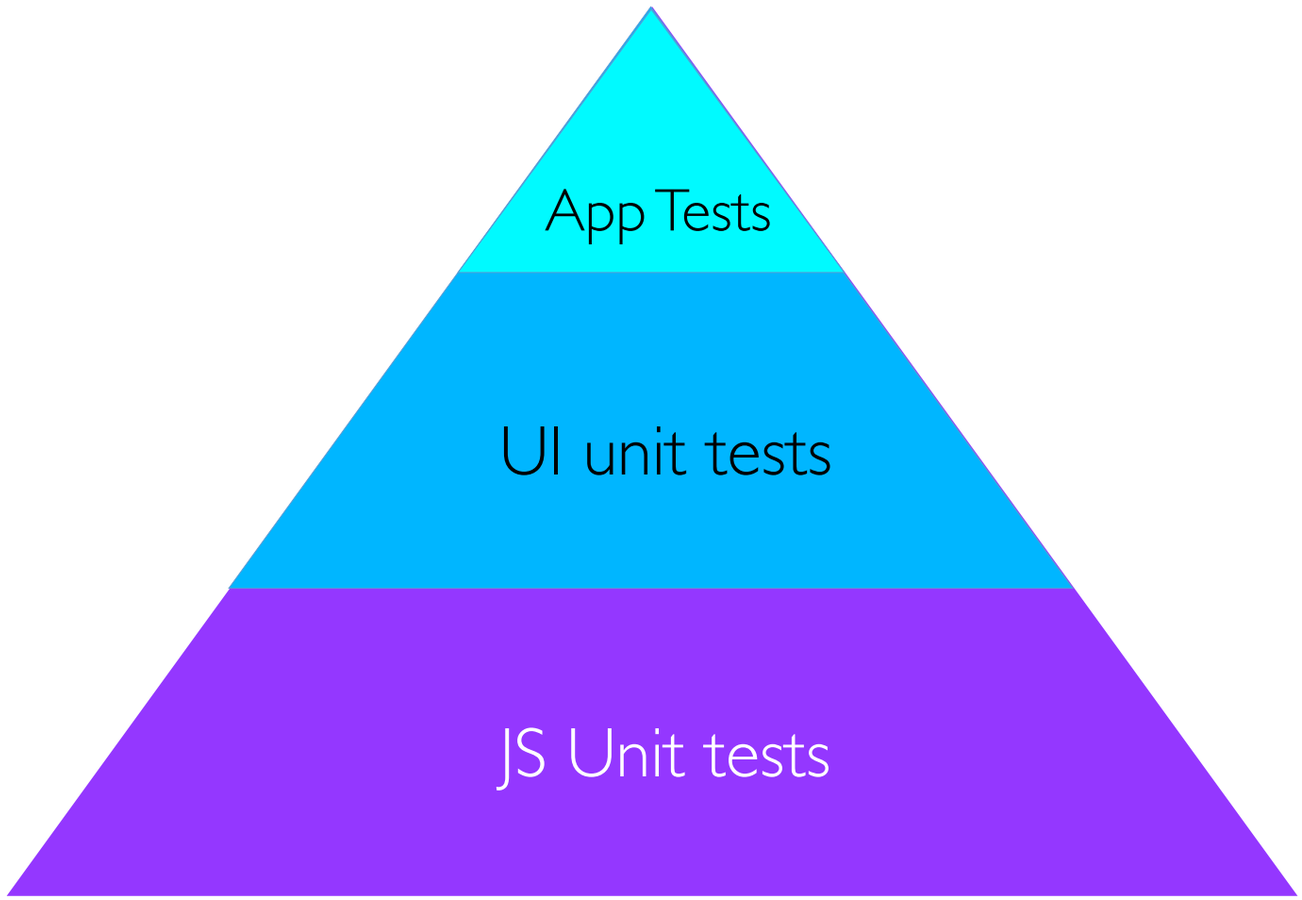
Application tests

- Siesta will visit a URL
- All application code will be executed
- Errors will be harder to find
- Still has high value, tells you if the application as a whole works or not

Best practices

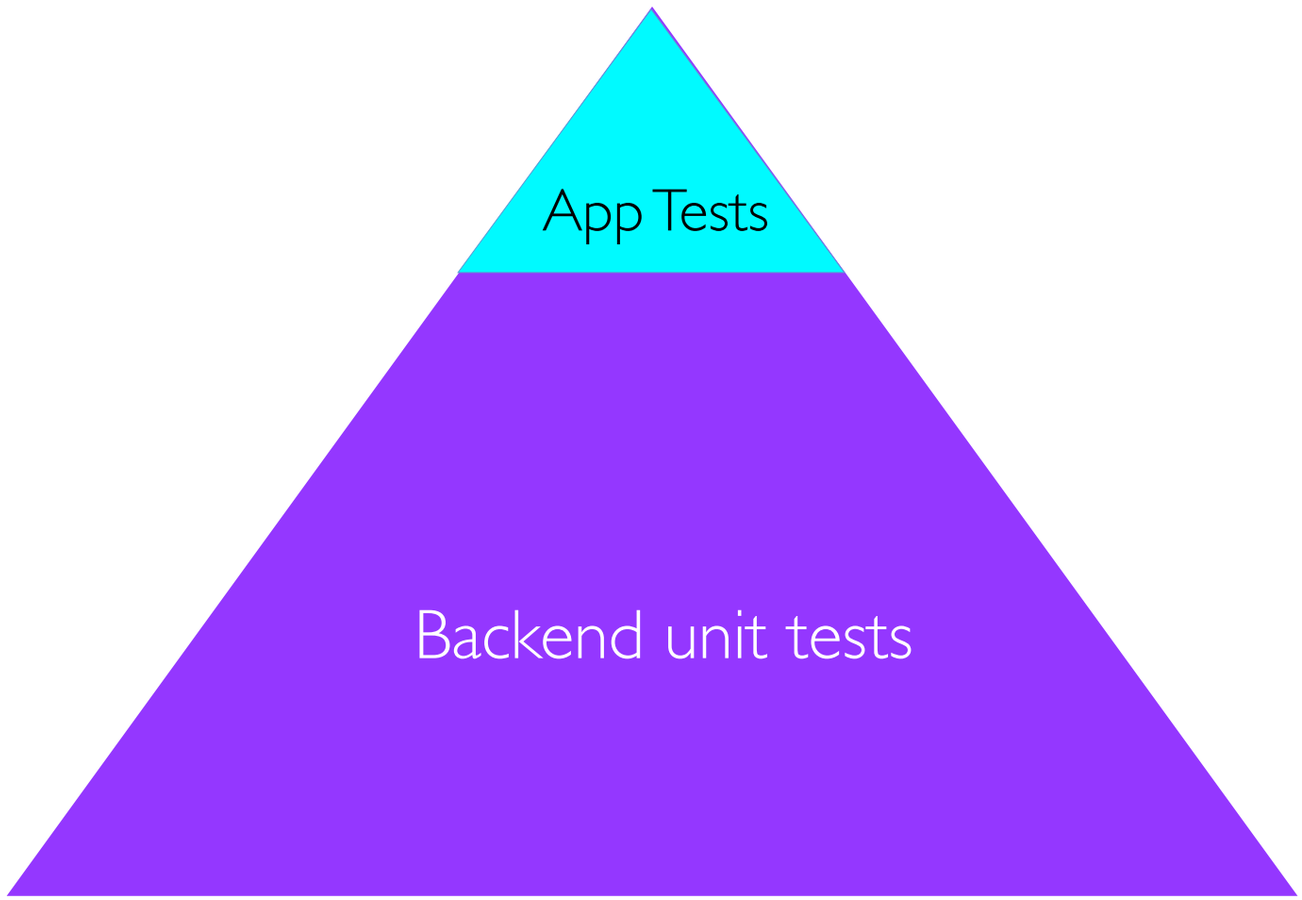
- Focus on unit tests first, test your model, logic, utility classes
- You can also test views in isolation, a *"UI unit test"*
- The less code executed in a test, the faster you will be able to find an error if a test fails
- Name test files "**xxx.t.js**" to not confuse them with application files

Ideal Test Pyramid



SENCHACOM MUNITYDAYS 19

Real World



Siesta Unit Tests

A Siesta unit test

- Is usually focused on pure logic, date lib, financial calc, Model, Store
- Focus on one class only
- The `StartTest` method is called as soon as all preloads are loaded and page is '*ready*'
- Errors are easy to find

Basic assertions

```
describe("Basic stuff", function(t) {
    t.is(1, 1, '1 is 1')
    t.isnt(1, 2, "1 isn't 2")

    t.like('Yo', /yo/i, '"Yo" is like "yo"')

    t.throwsOk(function () {
        throw "yo"
    }, /yo/, 'Exception thrown, and its "yo"')

    t.pass("Foo");
    t.fail("Did not expect this");

    t.isDeeply({ a : 1, b : 2 }, { a : 1, b : 2 }, 'Correct')
})
```

Strict type comparison

```
describe("Type comparison", function(t) {
    t.is(1, '1');         // Uses == PASS

    t.isStrict(1, '1'); // Uses === FAIL
})
```

Deep object comparison

```
describe("Object comparison", function(t) {
    t.is({ foo : 1 }, { foo : 1 });         // FAIL

    t.isDeeply({ foo : 1 }, { foo : 1 }); // PASS

    t.is([1,2,3], [1,2,3]); // FAIL

    t.isDeeply([1,2,3], [1,2,3]); // PASS
})
```

SENCHACOMMUNITYDAYS 19

A simple unit test

Project.js

```
Project.start(
    {
        group   : "Model tests",
        preload : [
            “ext-all.js",
            "user.js"
        ],
        items   : [
            "001_user.t.js"
        ]
    }
);
```

001_user.t.js

```
describe("User tests", function (t) {
    var user = new User({
        name : "Bob"
    });

    t.expect(user.name).toBe("Bob");
});
```

SENCHACOMMUNITYDAYS 19

Or describe test as

```
Project.start({
    group : 'Model tests',
    items : [
        {
            url          : '001_user.t.js',
            alsoPreload : ["foo.js"]
        }
    ]
});
```

TDD in practice

- Red
- Green
- Refactor
- Repeat

SENCHACOMMUNITYDAYS 19

TDD DEMO

Exercise

Objective:

- 1. Configure the Project with one test.
- 2. Test the Employee class at http://jsfiddle.net/bryntum/bmsqmybz/

SENCHACOMMUNITYDAYS 19

Questions?

BDD

Test suites & specs

BDD style testing focuses on making tests readable, which results in the test suite becoming a bit like a readable documentation.

A **test suite** means a container for **specs** or other test suites.

```
StartTest(function (t) {
    t.describe("My system", function (t) {
        t.it("Should allow user to log in", function (t) {
        })

        t.describe("Report engine of my system", function (t) {
            t.it("Should allow generate reports in PDF", function (t) {
            })
        })
    })
})
```

BDD / Subtests

In Siesta, we also refer to **it-**statements as subtests. For readability, debuggability and maintainability: group your assertions `t.it` and `t.describe`. Note that the function passed to **t.it** is passed the sub-test instance **“t”**.

```
describe("User model tests", function (t) {

    t.it("Nested test here", function (t) {
        var user = new User({ name : "Bob" });

        t.expect(user.name).toBe("Bob");
    });
});
```

BDD / Subtests

You can nest test groups infinitely.

```
describe("User model tests", function (t) {

    t.it("Nested test here", function (t) {

        t.it("Nested test here", function (t) {

            t.describe("Nested test here", function (t) {

            });
        });
    });
});
```

iit / xit

`iit` to isolate and run only a certain test group. `xit` to skip.

IMPORTANT: Don't allow checking in tests containing `iit`

```
t.iit("Isolated test", function(t) {
});

t.it("Ignored test due to iit", function(t) {
});

t.xit("Skipped test", function(t) {
});
```

beforeEach / afterEach

- Hooks that allow you to repeat certain pieces of code before or after each subtest
- + Helps you keep your tests as small as possible
- - DRY isn't everything, prioritise readability.

beforeEach / afterEach

```
describe('Some spec', function (t) {

    var user;

    t.beforeEach(function() {
        user = new User();
    });

    t.afterEach(function() {  });

    t.it('Should test something', function (t) {
    });
});
```

Exercise

Objective:

- 1. Convert the tests you wrote for the Employee class to BDD style
- 2. Wrap assertions in t.it, and give a useful description. Use beforeEach to instantiate your test subject.
- 3. Try **t.iit** too

SENCHACOMMUNITYDAYS 19

Questions?

Utility methods

todo / snooze

- Sometimes allowing a test to temporarily fail is useful
- You have time to write the test, but not the fix
- Postpone the fix but keep monitoring the bug using the test

todo / snooze

```
t.todo(function(t) {…});

t.snooze("2019-01-05", function(t) {…});

t.knownBugIn(“6.7.0", function(t) { … });
```

SENCHACOMMUNITYDAYS 19

Demo todo tests

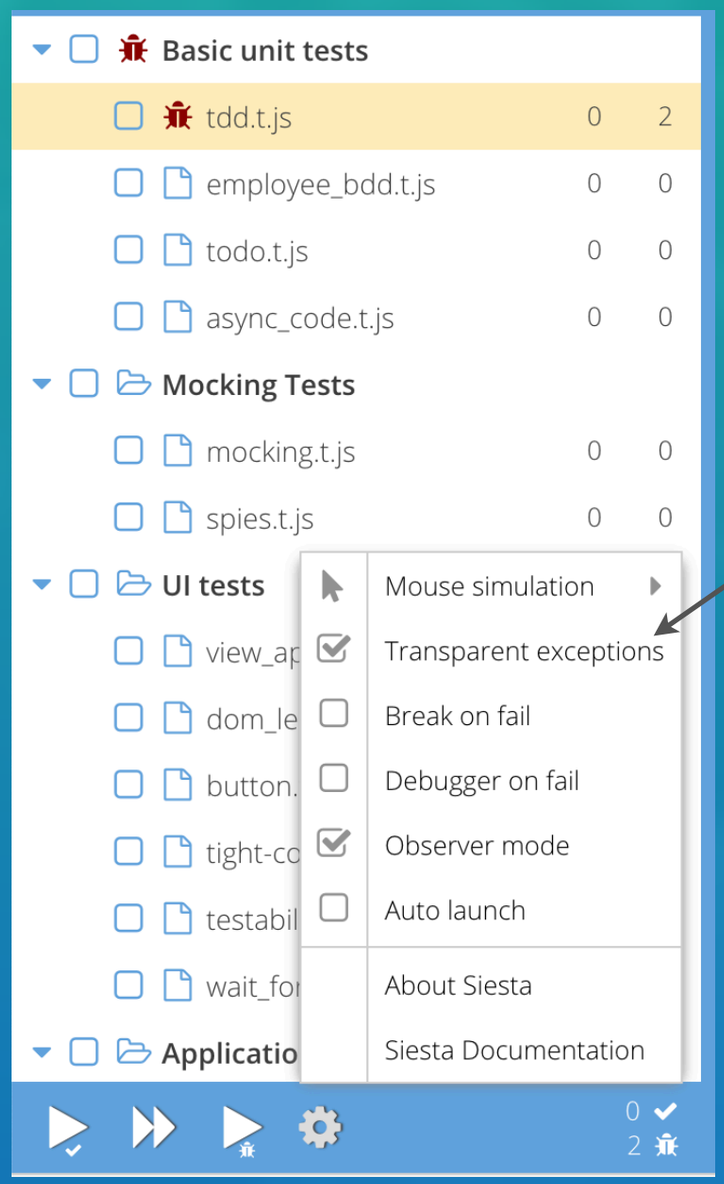
t.diag

- Outputs a diagnostics message in the assertion list
- Useful for test documentation, debugging

```
describe("User model tests", function (t) {

    t.diag("Some message here”);
});
```

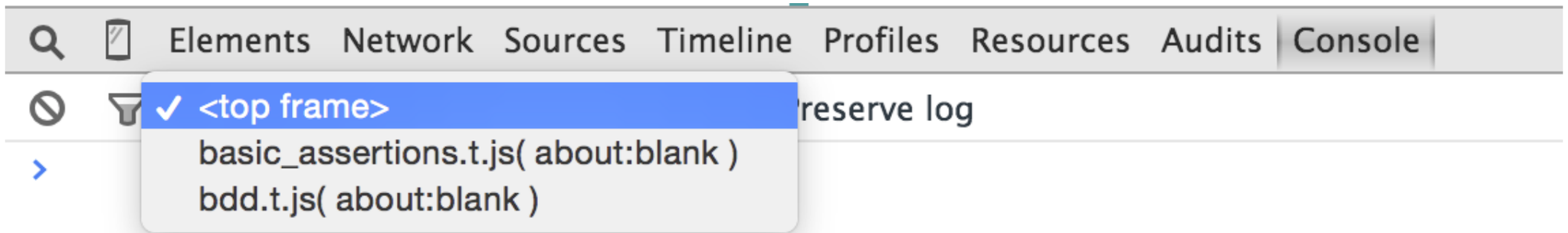
Debugging tests

Make errors bubble to console

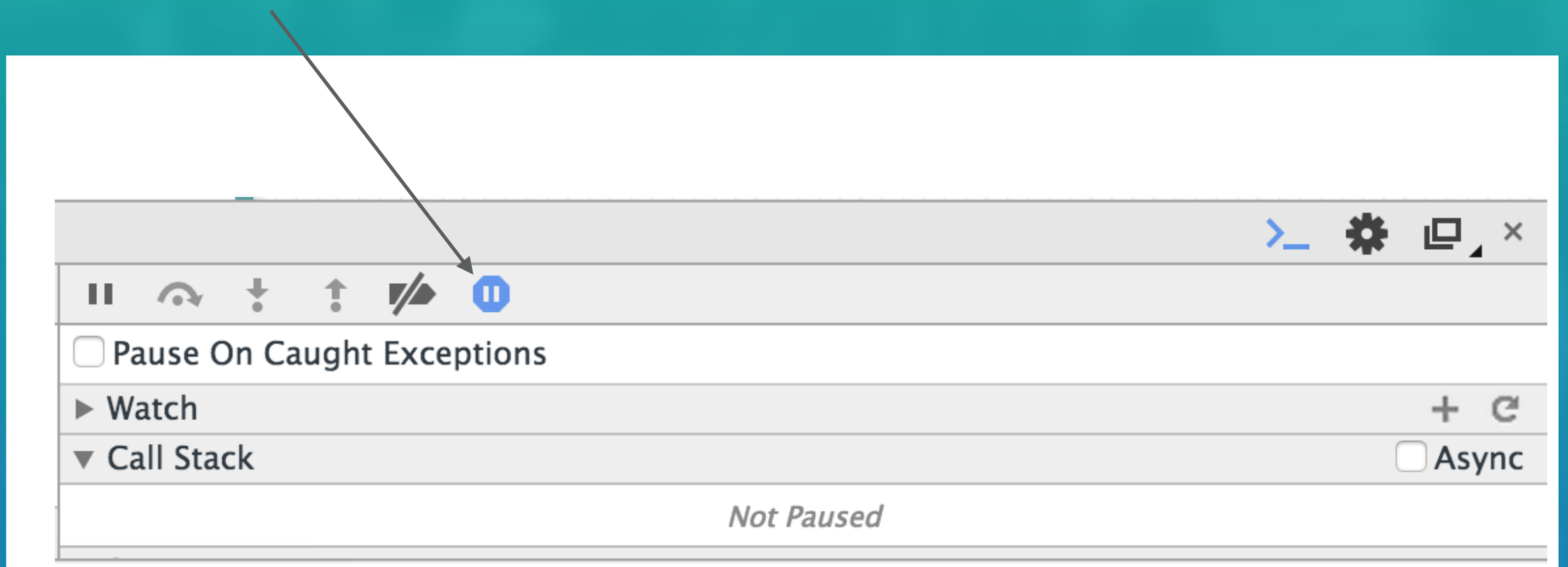


Transparent exceptions => errors bubble

Pick the right frame



Dev tools: break on exceptions



Keyboard shortcut

CTRL-E to rerun last test

Debugging exercise #1

Objective:

1. Set a **debugger;** breakpoint in your test code and make sure you can step line by line through the test

Debugging exercise #2

Objective:

1. Make sure you can see the Employee class of the last exercise

```
> Employee
<· function g(CacheBuster)
> |
```

Debugging exercise #3

Objective:

1. Add a typo to your test (misspell some a variable)
2. Make Chrome break and stop on an error (enable transparent exceptions)

SENCHACOMMUNITYDAYS 19

Questions?

Testing async code

Testing async code

Many times you'll find you need to wait for some condition.

In such a situation, you need to tell Siesta that an async flow is in process (to prevent Siesta from finalizing the test too early).

```
// Starts an "async code frame"
var as = t.beginAsync();

store.load(function () {
    // Do more test stuff
    t.endAsync(as);
});
```

Testing async code

```
t.wait(“foo”);

Ext.Ajax.request({
    url     : 'ajax_demo/sample.json',

    success : function() {
        // Do more test stuff
        t.endWait(“foo”);
    }
});
```

Demo async testing

Spies and mocking

Test spies

Asserts functions are called (or not called) as expected

```
t.isCalledOnce('purr', cat);
t.isCalled('purr', cat);
t.isCalledNTimes('purr', cat, 1);
t.isntCalled('putClawsOnFace', cat);
```

Test spies

Check arguments, return value, number of calls

Calls the original implementation by default

```
t.spyOn(obj, 'someMethod').callThrough()

obj.someMethod(0, 1)
obj.someMethod(1, 2)

t.expect(obj.someMethod.calls.any()).toBe(true)
t.expect(obj.someMethod.calls.count()).toBe(2)
t.expect(obj.someMethod.calls.first()).toEqual({
    object      : obj,
    args        : [0, 1],
    returnValue : undefined
})

t.spyOn(obj, 'someMethod').and.callThrough()
```

Demo spies

Mocking server ajax responses

- Dealing with database state in application tests can be painful

- Either backup/restore DB to a known state. Slower, but gives you a full-stack test of your entire application.

- …or mock the server responses. Faster, but can be time consuming if DB / server API changes frequently.

Mocking server ajax responses

- Mocking support exists in the Ext JS SDK (/examples/ux)
- Open Ext JS docs, see the Ajax `Simlet` class
- Example found in the siesta/examples folder
- http://www.bryntum.com/blog/mocking-ajax-calls-with-siesta/

Exercise: Ajax mocking

Objective:

1. Create a basic Ext JS store with an Ajax proxy
2. Inject a mocked JSON-simlet and assert that it loads

Testing views

View tests

- More complex by nature, as this involves DOM
- Useful to do "UI unit tests" of your larger views
- Functional testing
  - User interactions
  - Waiting
  - Asserting
- Test UI components in isolation, reveals tight coupling.

SENCHACOMMUNITYDAYS 19

View API test

Custom LoginForm

```
Ext.define('LoginForm', {
    extend      : 'Ext.FormPanel',
    title       : 'Login',
    defaultType : 'textfield',
    loggedIn    : false,

items : ['{
    fieldLabel : 'Name',
    name       : 'name'
}, {
    fieldLabel : 'Password',
    inputType  : 'password'
}],

/**
 *
 * @returns {boolean}
 */
isLoggedIn : function () {
    return this.loggedIn;
},

/**
 *
 * @param {boolean} value
 */
setLoggedIn : function (value) {
    this.loggedIn = value;
},
```

View API test

Test public API methods of a custom LoginForm

```
describe('Login form', function (t) {
    var form;

    t.beforeEach(function() {
        form && form.destroy();

        form = new LoginForm({
            id  : 'my-form',
            cls : 'some-css-class'
        });
    })

    t.it('Should be able set/get if user is logged in', function (t) {

        t.expect(form.isLoggedIn()).toBe(false);

        form.setLoggedIn(true);

        t.expect(form.isLoggedIn()).toBe(true);
    });
});
```

SENCHACOMMUNITYDAYS 19

Demo UI Unit test

DOM leaks

- Also make sure views don't leak DOM nodes or Components
- If for example a view creates a custom tooltip, remember to clean it up after the view is destroyed.

SENCHACOMMUNITYDAYS 19

Demo DOM leaks test

The `chain` method

The `chain` command

Since most UI testing scenarios are async, we use `t.chain` to push commands onto an asynchronous queue.

Each "**step**" in the queue can be an *object* or a *function*.

Functions receive a continuation callback as the first argument

The `chain` command

```
t.chain(
    { action : "click", target : "#foo" },
    { click : ".foo" },

    function (next) {
        // Any code can go here, remember to call next
        next();
    },

    function (next) {
        // Advancing the chain by passing the callback to another method
        t.click("#foo", next);
    }
);
```

Simulating user inputs

Simply use the various chain actions to interact with the UI.

```
t.chain(
    { click : "#someinput" },
    { type : "Mike[TAB]Password[ENTER]" },

    function (next) {
        // Assert that login was successful
        next();
    }
);
```

Targeting the DOM

- Siesta supports many ways of targeting the DOM content.

- Generic HTML website

  - CSS selector (‘#foo’, ‘body > .login-button’)
  - DOM Element reference
  - Coordinate (avoid)
  - A function returning any of the above

Targeting the DOM

- Ext JS application
  - Ext JS component or Ext.Element
  - Component query selector
  - Composite query selector
- You can also provide an ‘offset’ to click exactly at a certain point within your target (generally avoid)

Specificity matters

- If your selector isn't specific enough it could match many targets. In this case the first target will be used and a warning will be added.
- This can be very confusing, always make sure you target a single element.
- For CQ, try to decorate your components with a unique id or a custom attribute.

DOM / Ext Element

```
// HTMLElement
var el2 = document.getElementById("foo")

// Ext Element
var el = Ext.getBody()
```

Selectors

// CSS

“div.foo”

“#someId”

// Component Query, prefixed by >>

“>> textfield”

“>> somextype[foo=bar]”

// Composite Query

“gridpanel => .x-grid-cell:nth-child(2)”

Exercise

Objective:

- 1. Use t.chain and click in the center of the document.body
- After click, put a function with a **debugger;** statement

Selectors

// Composite Query

“gridpanel => .x-grid-cell:nth-child(2)”

“panel:not([hidden]) => .someCls”

// Targeting nested iframes

“#idOfFrame -> .some-selector”

Special CSS Selectors

```
// Targeting based on text content
"label:contains(Name)"
".someCls:textEquals(Age)"
```

SENCHACOMMUNITYDAYS 19

Targeting options

```
var myGrid = new Ext.GridPanel({ id : "myGrid" });

t.chain(
    { click : [10, 10] },
    { click : "#myGrid" },
    { click : myGrid },
    { click : doc.getElementById("myGrid") },
    { click : myGrid.getEl() },
    { click : ">> gridpanel", offset : ["50%", 10] },
    { click : "gridpanel => .x-grid-row" },
    {
        click : function () {
            return myGrid.el;
        }
    }
);
```

Exercise

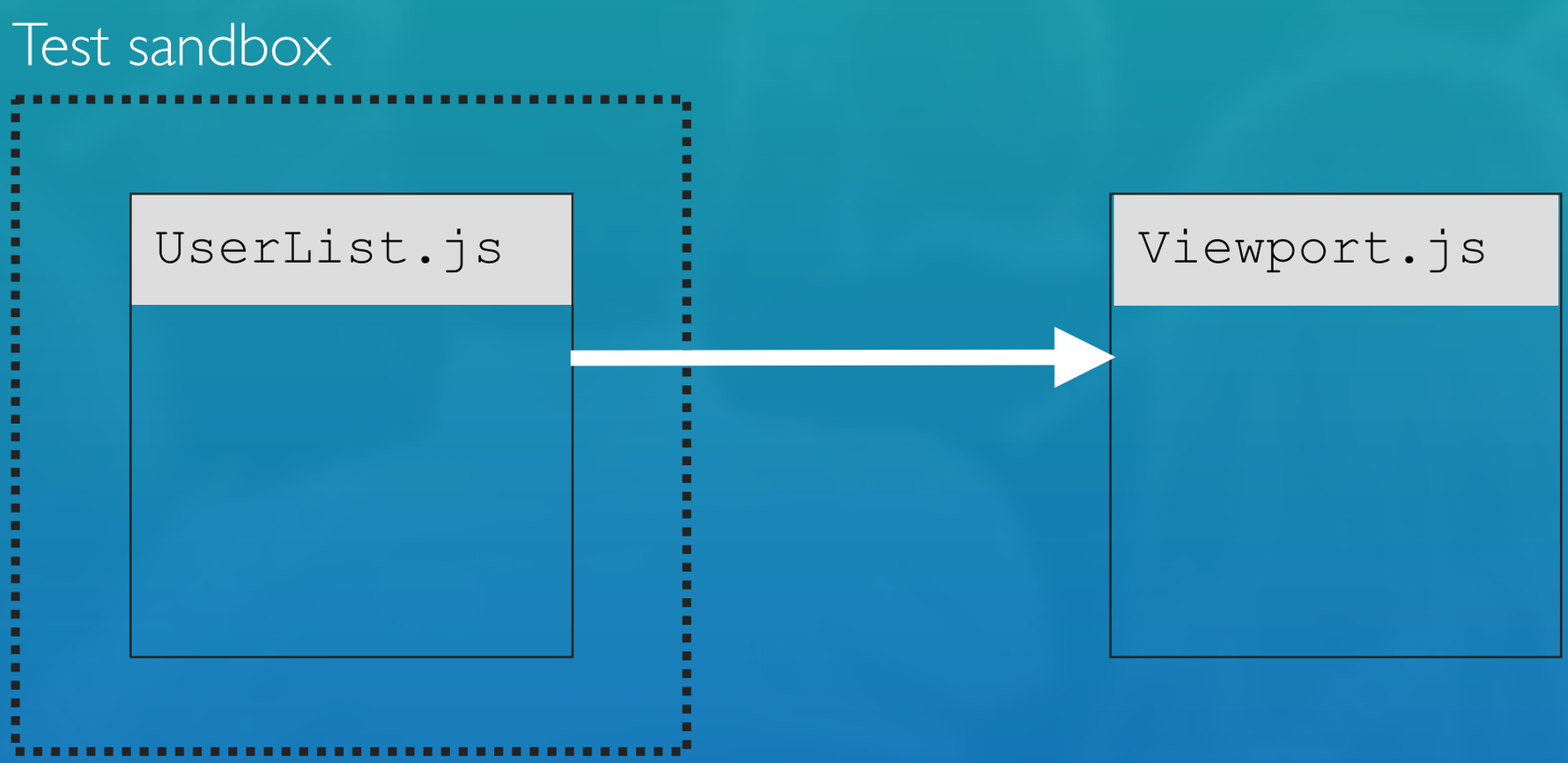
Objective:

- 1. Render an Ext JS Button in a test
- 2. Use t.chain and click it in as many ways you can
- 3. Assert the number of clicks made (`t.willFireNTimes`)

Synthetic JS event limitations

- Double clicking text doesn't select it
- Clicking link with #foo doesn't do pushState of history
- Cannot interact with native context menu, file upload, alert, confirm, prompt, print dialog, auto-complete values, scrollbars
- Synthetic 'mouseover' events, :hover CSS rules aren't triggered
- Cannot interact with frames on another domain
- Siesta also supports native OS level events

Tight coupling



SENCHACOMMUNITYDAYS 19

Demo tight coupling

Waiting

Waiting

- With UI tests, you often need to wait:
- Grid : waitForRowsVisible
- Data : waitForStoresToLoad
- CSS : waitForSelector
- 40+ waitForXXX commands in the API

Waiting

```
t.chain(
    { waitForSelector : "#someinput" },
    { waitFor         : 1000 },   // Avoid
    { waitForEvent    : [document.body, "click"] },

    function (next) {
        // Programmatically wait
        t.waitFor(1000, next);
    }
);
```

waitFor : function() {...}

You can also wait for any condition using a callback

```
t.chain(
    {
        waitFor : function() {
            return window.foo == 1;
        }
    },
    ....
);
```

Wait for…

- You can control the waiting timeout, default: 10s. In a chain you run waitFor commands using:
- `{ waitForSelector : '.foo', timeout : 30000 }`
- Find all wait commands in the docs as `waitForXXX`.

Animations

- Animations can introduce tricky race conditions in your UI tests
- Easiest solution is to turn animations off for your app globally
- Or add `waitForAnimations` statements where needed

Wait for animations

```
t.chain(
    { click : ">>#somePanel tool[type=expand-bottom]" },

    //{ waitForAnimations : [] },

    {
        click : ">>panel button"
    }
);
```

SENCHACOMMUNITYDAYS 19

waitForAnimations demo

Wait with trigger

Sometimes you need to trigger a wait along with an action to avoid race conditions

```
// Naive test implementation
t.chain(
    { click : ".loginbutton" },
    { waitFor : “PageLoad" },
    ...
);


// Better
t.chain(
    {  waitFor : "PageLoad", { trigger : { click : ".loginbutton" } },
    ...
);
```

Exercise

Objective:

- 1. Launch a custom waitFor command, detecting the presence of a DIV with content: Hello on the page: Hello

- 2. Inject a DIV with text "Hello" delayed using setTimeout

Writing testable code

Testable code

- Put logic where it's easy to test
- Be open to changing your code patterns for testability
- Test your stable public API, avoid testing private internals

SENCHACOMMUNITYDAYS 19

WHAT SHOULD WE TEST?

Models & Stores

- Pure JS, not tied to DOM
- Testing is easy
- Good place to put business logic
- Tests are a great investment

Views

- View produces HTML, involves DOM
- Testing requires more effort. Render. Interact.
- Bad bad place to put business logic

Controllers

- Not directly tied to DOM
- But, often tied to components via CQ
- Hard to isolate, mocking dependencies not trivial
- Easier tested through application tests

Application

- "Black box testing"
- Launch app UI, issue commands to browser
- Does application work or not?

SENCHACOMMUNITYDAYS 19

Don’t test the framework

Your tests shouldn’t test core code of Ext JS

```
t.is(myModel.get("name"), "Mike", "Model.get works");
```

Testability demo

Application tests

Application tests

- Tell Siesta to go to a URL, index.html (on same domain)
- No need to preload anything, black box testing
- Wait for page to load, simulate a user to login and interact with the application

Application test challenges

- More complex by nature
- Usually DOM heavy + lots of user interaction
- Need to manage DB, put it in a known state
- Handle session
- Tests can become fragile unless careful
- Harder to find errors

An application test

```
Project.start(
    {
        group       : "Application tests",
        pageUrl     : "index.html",
        items       : [
            "001_login.t.js"
        ]
    }
);
```

Exercise

Objective:

1. Open one of the Ext JS example applications
2. Make it open in a test with the Siesta pageUrl option

Exercise

Objective:

1. Continue with the test of the Ext JS example
2. Write 3 chain steps to interact with the application

Dealing with native dialogs

- `alert/confirm/prompt/print` all block the main UI thread, will break tests.
- Siesta therefore replaces these native dialogs to avoid tests from freezing

```
t.setNextConfirmReturnValue(false);

confirm('foo'); // Returns false
```

Taking screenshots

Note: Only supported when launched from WebDriver

```
describe("Screenshots", function (t) {

    t.chain(
        { waitForTarget : "panel => :(Grid Window)" },
        { screenshot    : "filename" } // “filename.png”
    );
})
```

Demo screenshots

```
puppeteer lh/training/SenchaCommunityDays-SiestaWorkshop/examples/ --include screenshot.t.js
```

Cross page testing

- Sometimes application tests navigate between pages
- This has implications on the test code, as it runs in the target page by default
- Set the **enablePageRedirect** config for such test scenarios
- Note that Siesta is limited by *Same-Origin-Policy,* can't test URLs on other domains.

Cross page testing

- In a cross page test, in the test code, **`window`** will now point to the top Siesta window
- Get access to the test window via **`t.global`**
- This means any global variables need to be read from the test global, including **`Ext`**.

CROSS PAGE TESTING

```
{
    group                : 'Application Tests',
    enablePageRedirect   : true,   // For apps that navigate to different page URLs
    items                : [
        {
            pageUrl      : 'application/app.html',
            url          : 'application/app.t.js'
        }
    ]
}
```

SENCHACOMMUNITYDAYS 19

CROSS PAGE TESTING

```
describe('Cross page testing', function (t) {

    t.chain(
        { click : '.loginbutton' },

        // First wait for page to be ready
        { waitFor : 'PageLoad' },

        // Now wait for application to be ready
        { waitForCQ : 'viewport > panel[rendered=true]' },

        function() {
            // Access window from the test instance
            var win = t.global;
            var Ext = t.getExt();
            var panel = Ext.ComponentQuery.query('viewport > panel')[0];

            t.expect(panel.title).toBe('The application')
        }
    )
})
```

SENCHACOMMUNITYDAYS 19

Demo cross-page testing

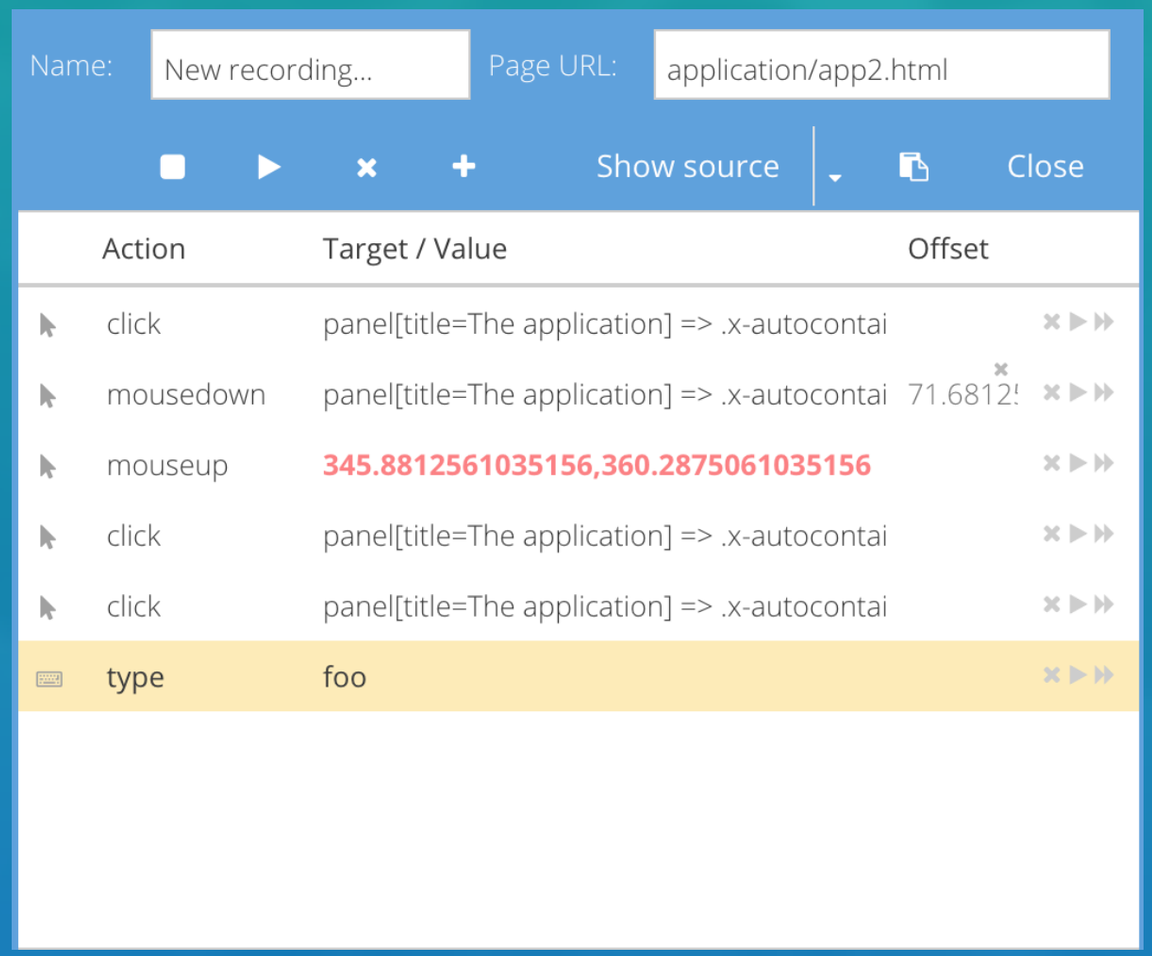
Event recorder

When to use the recorder

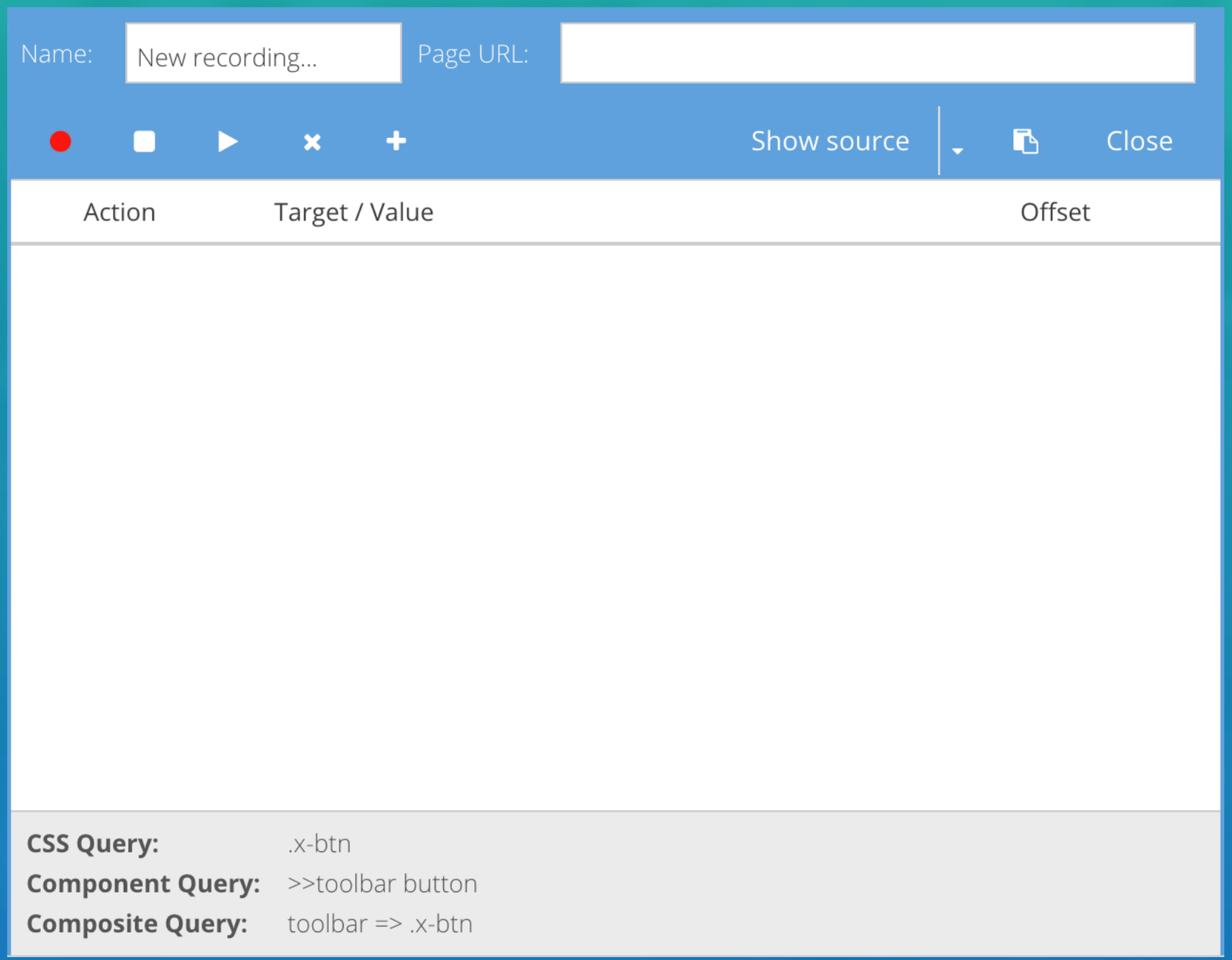
- To record full application test scenarios

or

- To record a view integration test - first create fixture in a test JS file and record the interactions to save time.



Recorder buttons



Recording UI events

- Actions & targets can be modified after recording
- Actions, offsets and targets should be reviewed
- When done, export script to JS and put in a test file
- Add test descriptor to your suite - `Project.start()`

Recording UI events

- By default, the recorder keeps offsets for each action to be able to click at the exact same location
- Most likely you can discard the offsets and target the center coord.
- Set `recordOffsets` to false to skip offsets.
- http://bryntum.com/docs/siesta/#!/api/Siesta.Recorder.Recorder-cfg-recordOffsets

Recorder limitations

- Cannot interact with native dialogs:
  - *alert, prompt, confirm, file upload* etc.
- Record clicks native scrollbars
- Interacting with the native browser window
- Only in "good" browsers

SENCHACOMMUNITYDAYS 19

Recorder demo

Locating components

- Recorder will try to find the most stable Ext JS CQ selector
- For Ext JS components, the following attributes are used

```
'id'
'itemId',
'text',          // menu items, buttons
'dataIndex',     // Column component
'iconCls',       // button/menuitem
'type',          // Panel header tools
'name',          // form fields
'title'          // identifying panels, tab panels headers
```

Recorder configuration

You can override and provide custom attributes to use. See `recorderConfig` config param in Project docs.

```
/**
 * @cfg {Array[String]/String} uniqueComponentProperty A string or an array of strings, containing attribute names that the Recorder will use to identify Ext JS components.
 */
uniqueComponentProperty : null,

/**
 * @cfg {String} uniqueDomNodeProperty A property that will be used to uniquely identify DOM nodes.
 */
uniqueDomNodeProperty : 'id'
```

Exercise

Objective:

- 1. Open an Ext JS 6 example (Executive Dashboard)
- 2. Record a test script that clicks all the tabs in the left side
- 3. Polish, export, put it in a test file, add to your Project

TIPS N TRICKS

If your application doesn't have well decorated components (id or itemId set), then here's an easy override to make components testable without too much effort.

```
Ext.Component.override({
    onRender : function () {
        this.callParent(arguments);

        this.el.set({
            TEST_ROLE : this.TEST_ROLE || this.$className
        })
    }
});
```

Extending Siesta

Siesta Test Class

- As your test suite grows, you may find the same code repeated in multiple tests
- Time to refactor and break that functionality out into its own Siesta TestClass.
- Typical case: several of your application tests require a Login to happen before running the test

Typical application test

```
describe("Login test", function(t) {
    t.chain(
        { click : '>> loginfield' },
        { type : 'john[TAB]' },
        { type : 'secret[ENTER]'},
        { waitForPageLoad : [] }
    );
});
```

Siesta internals

- Siesta is built upon the Joose JavaScript class framework.
- Easy to create and compose classes using inheritance and mixins.
- http://joose.github.io/Joose/doc/html/Joose/Manual.html

Joose subclasses

```
// Define a class
Class('YourClass', {
    // extend via “isa”
    isa: TheSuperClass,

    // methods are placed in a “methods” object
    methods: {
        someFn     : function(callback) { ... },
        assertThat : function(msg) { ... }
    }
});
```

Joose class attributes

```
Class('YourClass', {
    // extend via “isa”
    isa: TheSuperClass,

    // attributes are placed in a “has” object
    has: {
        loggedIn     : false,
        someArray  : Joose.I.Array,
        nbrUsers     : 0
    }
});
```

Joose roles

```
// Use a Role to define attributes and methods that can be added to any class
Role('Login', {

    // attributes are placed in a “has” object
    has: {
        loggedIn     : false,
        someArray  : Joose.I.Array,
        nbrUsers     : 0
    },
    methods : {
        doLogin : function() { ... }
    }
});
```

Consuming a Role

```
Class('YourClass', {
    does : [
        Login
    ]
});
```

Creating your own test class

- The Project can be configured to use a TestClass, which provides additional assertion methods or utility functions.
- By default, Siesta.Project.Browser.ExtJS is configured to use Siesta.Test.ExtJS.
- **Important:** Include your test class JS file after Siesta in your HTML page.

Refactoring test, step # 1

```
Class('Your.Test', {
    isa: Siesta.Test.ExtJS,

    methods: {
        login : function(user, pw, callback) {
        this.chain(
          { type : user, target : '>> loginfield' },
          { type : pw +'[ENTER]', target : '>> passwordfield' },
          callback
        );
     }
  }
});
```

REFACTORING TEST #2

```
describe(“Login test”, function(t) {
    t.login(“Bob”, “Dylan”, function() {
        // Now we’re logged in
    });
});
```

HARNESS HTML PAGE

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="ext-all.css">
        <link rel="stylesheet" type="text/css" href="../resources/css/siesta-all.css">

        <script type="text/javascript" src="ext-all-debug.js"></script>
        <script type="text/javascript" src="../siesta-all.js"></script>
         <script type="text/javascript" src="Your.Test.js"></script>
        <script type="text/javascript" src="index.js"></script>
    </head>
    <body></body>
</html>
```

HARNESS

```
Project.configure({
    testClass : Your.Test,
    ...
});
```

TEST CLASS CAVEATS

The context of the code in your test class will be that of the ‘top’ window. To access the test iframe window, use ‘this.global’.

Same goes for Ext, if you type Ext it’ll refer to the Siesta UI copy of Ext JS. Instead use this.getExt().

Same as writing this.global.Ext

TEST CLASS CAVEATS

```
Class('Your.Test.Utilities', {
    isa : Siesta.Test.ExtJS,

    methods : {

            someFn : function () {
              var win = this.global;
                  var Ext = this.getExt();
            }
    }
});
```

EXERCISE

Objective:

1. Write your own test class which exposes a textIsPresent method

2. Write a simple test which calls this method

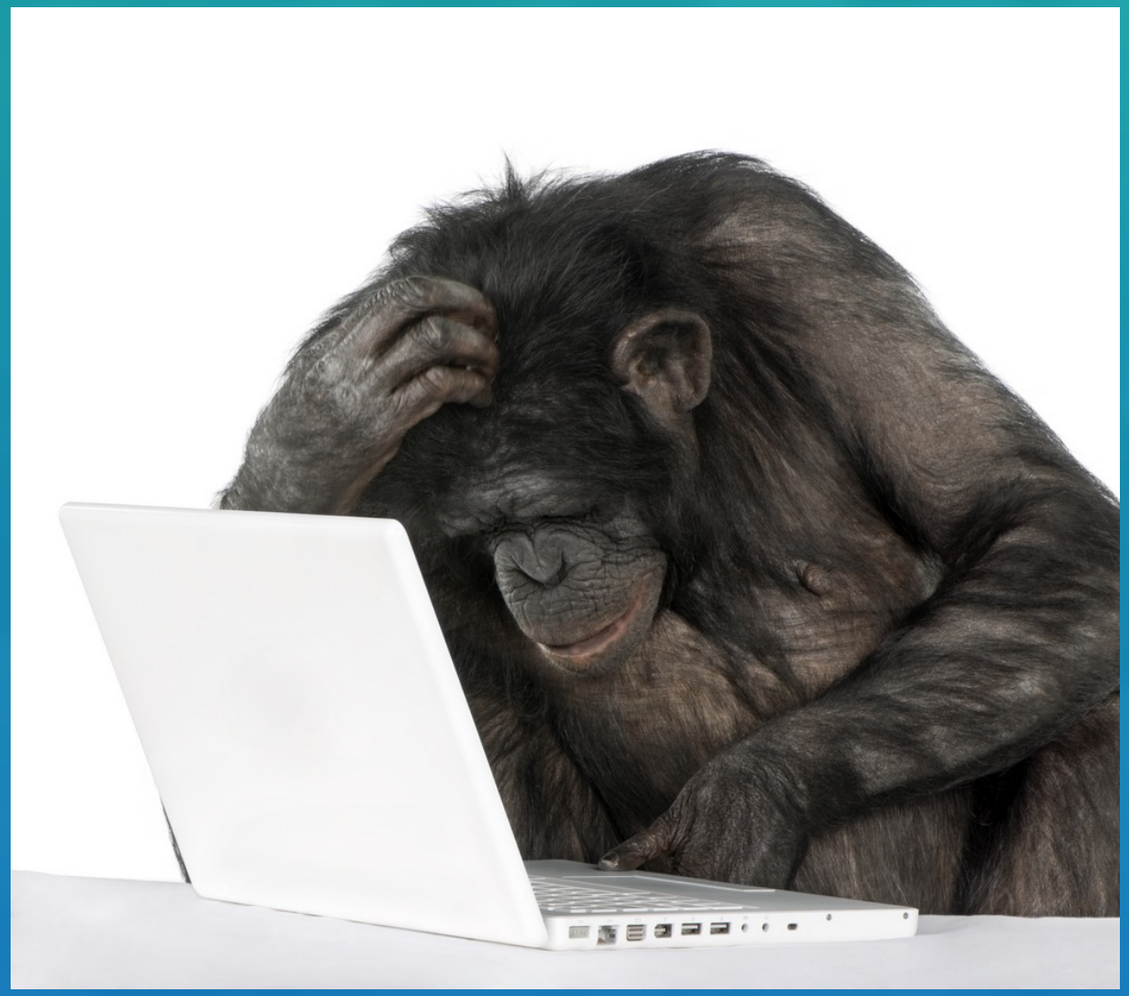
EXERCISE

```
Class('Your.Test.Utilities', {

    isa : Siesta.Test.ExtJS,

    methods : {

            textIsPresent : function (text, descr) {
             // Implement custom assertion here
            }
    }
});
```

Monkey tests

Monkey testing

- Random UI testing
- Clicks, types, drags etc. in your UI
- Finds unhandled exceptions
- Free testing help. 0$



Monkey testing

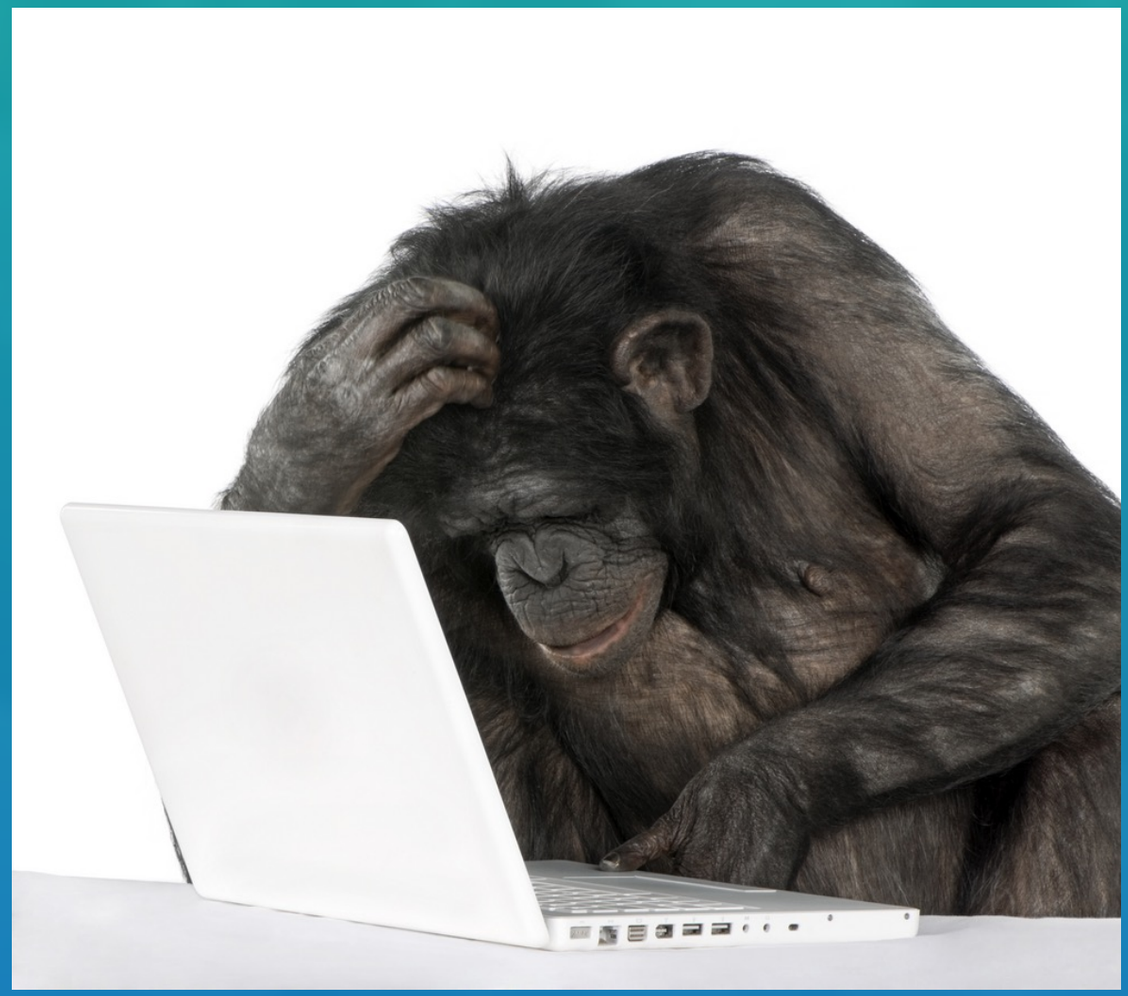
```
t.monkeyTest('>>gridpanel', 30)

t.monkeyTest('#someDiv', 30)

t.monkeyTest('body')   // Defaults to 10 actions
```

SENCHACOMMUNITYDAYS 19

When monkeys find something

- If there is an unhandled exception, the test logs all the actions done by the virtual monkey.

- This log can be directly pasted into a `t.chain` call to reproduce it easily.



Monkey testing demo

Code quality tests

Code quality tests

- If you have a large or a distributed team, achieving and maintaining a high quality code base is challenging, especially with JavaScript.
- It's easy to write tests to help you catch simple errors early before other developers are affected.
- Bryntum uses a lot of Component sanity check tests.

List of sanity tests

https://github.com/matsbryntse/Component-maker-test-pack

1. Your component can be loaded on demand with Ext.Loader (using ext-debug.js)
2. Your component doesn't create any global Ext JS overrides
3. It passes basic JsHint rules (no syntax errors, trailing commas, debugger)
4. It does not use global style rules ('.x-panel' etc)
5. It can be sub-classed
6. It does not leak any additional components or DOM elements
7. It doesn't put null/undefined garbage in the DOM
8. It doesn't override any private Ext JS methods in your component superclasses
9. It can be created, destroyed with and without being rendered first
10. It passes a basic monkey test (random interactions with it)
11. No layouts suspended, no garbage left in the DOM

Automation

SENCHACOMMUNITYDAYS 19

Automation

- Automation package is cross platform (Win, Mac, Linux).
- Siesta tests can be automated using either Puppeteer, SlimerJS, xvfb or Selenium WebDriver
- For in-IDE use, we recommend Puppeteer
- For nightly builds, UI tests and CI, we recommend WebDriver.
- WebDriver requires that the browsers to be tested are installed on the host

SENCHACOMMUNITYDAYS 19

Puppeteer

scheduler — node ◂ node scripts/build.js run tests — 140×36

```
[mankz:scheduler mats$
[mankz:scheduler mats$ scripts/build.js run tests
 Launching test suite, OS: MacOS, agent: Chrome 71.0.3563.0
 [PASS]  datalayer/023_loading_data.t.js
 [PASS]  datalayer/024_event.t.js
 [PASS]  datalayer/025_eventstore.t.js
 [PASS]  datalayer/026_custom_model_fields.t.js
 [PASS]  datalayer/026_customizable.t.js
 [PASS]  datalayer/027_resourcestore.t.js
 [PASS]  datalayer/028_timeaxis_dst.t.js
 [PASS]  datalayer/030_timeaxis_dst_chile.t.js
 [PASS]  datalayer/031_timeaxis_noncontinuous.t.js
 [PASS]  datalayer/032_timeaxis_date_methods.t.js
 [PASS]  datalayer/034_event_multi_assignment.t.js
 [PASS]  datalayer/035_id_consistency_manager.t.js
 [PASS]  datalayer/037_customizable_nested_set.t.js
 [PASS]  datalayer/AssignmentStore.t.js
 [PASS]  datalayer/ResourceTimeRangeStore.t.js
 [PASS]  datalayer/TimeAxisExclude.t.js
 [PASS]  datalayer/TimeAxisFiltered.t.js
 [PASS]  crud_manager/01_add_stores.t.js
 [PASS]  crud_manager/02_load_package.t.js
 [PASS]  crud_manager/02_load_package_filters.t.js
 [PASS]  crud_manager/03_change_set_package.t.js
 [PASS]  crud_manager/01_add_stores_proto.t.js
 [PASS]  crud_manager/01_add_stores_setCrudManager.t.js
 [PASS]  crud_manager/01_add_stores_storeIdProperty.t.js
```

Puppeteer

```
PATH_SIESTA/bin/puppeteer URL [OPTIONS]

> __SIESTA__/bin/puppeteer http://yourproject/tests/
index.html [OPTIONS]
```

Can also use **--include foo** to only run tests containing "foo" in the filename.

Webdriver

```
> __SIESTA__/bin/webdriver http://yourproject/tests/
index.html [OPTIONS]
```

By default launches tests in all available browsers.

Target a single browser by adding **--browser=chrome**

Exercise

Objective:

- 1. Launch one of the tests you wrote previously
- 2. Try using puppeteer

Webdriver gotchas

- Important. Using the WebDriver may require some manual configuration steps for IE.

- Please refer to: **http://code.google.com/p/selenium/wiki/InternetExplorerDriver**

- When running the test suite in IE, make sure the browser window is focused (on top of other windows) and mouse cursor is outside of the IE window.

Generating reports

- To output a report, provide the `--report-format option`
- Valid values are **JSON** or **JUnit**.
- For Puppeteer you must also provide `--report-file` (the filename to write to)
- For Selenium you should provide `--report-file-prefix.`

```
puppeteer YourApp/tests --report-format JSON --report-file foo.json
```

JSON report


```
{
    "testSuiteName": "Siesta self-hosting test suite",
    "startDate": 1343114314723,
    "endDate": 1343114315401,
    "passed": true,
    "testCases": [{
        "url": "010_sanity.t.js",
        "startDate": 1343114315390,
        "endDate": 1343114315396,
        "passed": true,
        "assertions": [{
            "passed": true,
            "description": "Siesta is here"
        }, {
            "passed": true,
            "description": "Siesta.Test is here"
        }, {
            "passed": true,
            "description": "Siesta.Project is here"
        }]
    }]
}
```

JUnit report

```
<testsuite errors="0" failures="1" hostname="localhost:8085" name="Ext Scheduler Test
Suite" tests="2" time="3.594" timestamp="2012-06-06T08:55:21.520">

  <testcase classname="Bryntum.Test" name="lifecycle/040_schedulergrid.t.js" time="1.238">
       <failure message="Oops" type="FAIL"></failure>
  </testcase>

  <testcase classname="Bryntum.Test" name="lifecycle/042_schedulergrid_right_columns.t.js"
  time="0.818">
  </testcase>
</testsuite>
```

Automation exit codes

0 - All tests passed successfully

1 - Some tests failed

2 - Inactivity timeout while running the test suite

3 - No supported browsers available on this machine

4 - No tests to run (probably filter doesn't match any test url)

5 - Can't open project page

6 - Wrong command line arguments

7 - Something was wrong (generic error code for Selenium, an error in one or more browsers)

8 - Exit after showing the Siesta version (when `--version` is provided)

Exercise

Objective:

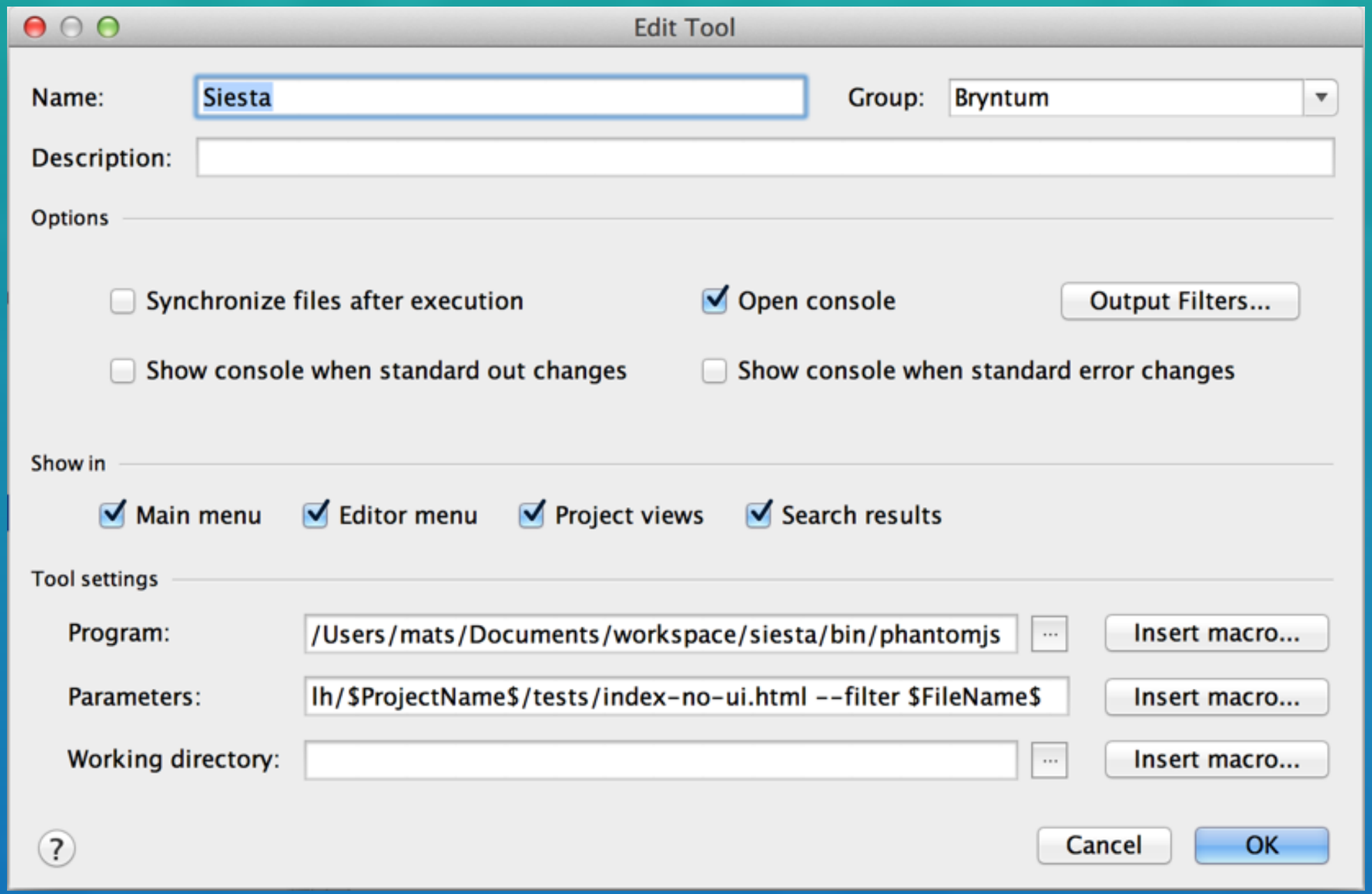
- Generate a basic JUNIT report using Puppeteer
- Generate a report using Chrome + Firefox with Webdriver

IDE integration

IDE integration

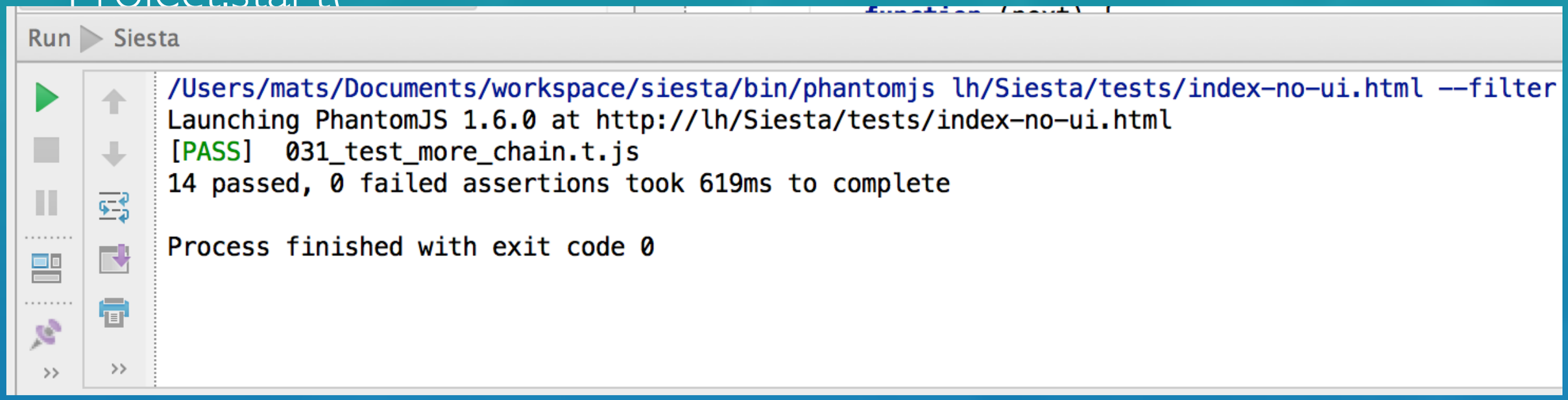
- Absolutely essential to not waste time
- Otherwise, to much time spent Alt+Tab to browser
- Very easy to setup in any decent IDE
- Bind to a one-hand key command for max speed

IDE integration



IDE output

- Project.start(



- items : [

SENCHACOMMUNITYDAYS 19

Integrates with…

- WebStorm
- Eclipse
- Visual Studio
- and many more...

Demo running test in IDE

Continuous Integration

SENCHACOM MUNITYDAYS 19

Purpose of CI

- Automated builds
- Nightly test suite execution
- Finding errors early => Code quality => Motivated developers
- Enables Continuous Delivery

ALWAYS READY TO RELEASE!

Integration

- Siesta integrates easily with all major CI tools:
- TeamCity
- Jenkins
- Bamboo
- Travis CI

Bryntum setup

- Bryntum uses TeamCity
- Test suites run on every commit in Chrome
- Nightly for all other browsers
- Reports, statistics, charts and code coverage

Pre-commit hooks

Pre-commit hooks are great for keeping codebase clean

Run your unit tests before commit, using a basic hook

Supported by all serious version control systems, **Git**, **SVN** etc.

.git/hooks/pre-commit

First step: running JSHint

```
# 1. Run JsHint
# ---------------------
files=$(git diff --name-only $CACHED --diff-filter=ACMRTUXB | grep '\.js$')

if [ "$files" != "" ]; then
    OIFS="$IFS"
    IFS=$'\n'
    for file in ${files}; do
        jshint "$file"

        if [ "$?" != "0" ]; then
            IFS="$OIFS"
            exit 1
        fi
    done
    IFS="$OIFS"
fi

echo "JSHint sanity test passed correctly"
```

.git/hooks/pre-commit

Now scan tests for **t.iit**.

```
# 2. Look for forgotten t.iit statements in tests
# -----------------------
git diff --name-only $CACHED --diff-filter=ACMRTUXB | while IFS= read -r line; do
    if [[ $line =~ ^tests\/.*\.t\.js$ ]]; then

        cat $line | grep -q '.iit('

        if [ $? == "0" ]; then
            echo "t.iit() statement found in file: $line"
            # will exit a pipe-subshell only, need additional check below
            exit 1
        fi
    fi
done

if [ "$?" != "0" ]; then
    exit 1
fi

echo "No t.iit statements found"
```

.git/hooks/pre-commit

Run some smoke tests

```
# 3. Run smoke tests
# ---------------------

# Build fresh version to run tests against
if [ -z $BRYNTUM_NO_NEED_TO_BUILD_BEFORE_COMMIT ]; then
    sencha build -p gantt.jsb3 -d .
fi


puppeteer http://lh/ExtGantt3.x/tests/index-no-ui.html?smoke=1 --pause 0 --no-color --page-size 100

exit $?
```

SENCHACOMMUNITYDAYS 19

Demo pre-commit hook

Cloud testing

Cloud testing

- Maintaining a VM farm is a lot of work
- Services like **Sauce** and **BrowserStack** solve this issue
- Run tests on any OS / browser combination
- Run in parallell for faster execution


```
bin/webdriver //localhost/ExtScheduler2.x/tests/index-no-ui.html
 --saucelabs USERNAME,KEY
--cap browserName=firefox
--include=010_sanity
--cap platform=XP
```

BrowserStack

SENCHACOMMUNITYDAYS 19

```
bin/webdriver //localhost/ExtScheduler2.x/tests/index-no-ui.html
 --browserstack USERNAME, KEY
 --cap browser=firefox
 --cap os=windows
 --cap os_version=XP
 --include=010_sanity
```

SENCHACOMMUNITYDAYS 19

Demo of cloud testing

Exercise

Objective:

- 1. Launch one of the tests you wrote previously in the cloud
- 2. Run it in Win 10, in Chrome using **Sauce**

Performance tips

Test sandboxing

- In general, a good idea since tests have a clean context
- Though comes with a high setup cost, due to preloading JS/CSS
- Sandbox setting can be disabled, use when large test groups share the same *preloads*.
- Be careful about polluting the global namespace.
- Up to 15-20x speed up for large test groups

SENCHACOMMUNITYDAYS 19

Demo sandboxing on/off

Remove the Siesta UI

When automating, use a special index-no-ui.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">

        <!-- Siesta application -->
        <script type="text/javascript" src="siesta-all.js"></script>

        <!-- Your test project -->
        <script type="text/javascript" src="index.js"></script>
    </head>
    <body></body>
</html>
```

More tips at....

www.bryntum.com/blog

Code coverage

What is code coverage?

- Measures % of code exercised by your tests.
- To be able to measure, code must first be instrumented.
- Siesta uses Istanbul for its code coverage feature.

Before instrumenting

```
var a = 1;
```

After instrumenting

```
if (typeof __coverage__ === 'undefined') { __coverage__ =
{}; }
    if (!__coverage__['filename.js']) {
       __coverage__['filename.js'] =
{"path":"filename.js","s":{"1":0},"b":{},"f":{},"fnMap":
{},"statementMap":{"1":{"start":{"line":1,"column":0},"end":
{"line":1,"column":10}}},"branchMap":{}};
    }
    var __cov_1 = __coverage__['filename.js'];
    __cov_1.s['1']++;var a=1;
```

Different coverage metrics

- Function coverage
- Branch coverage
- Line coverage
- Statement coverage

Useful metric..?

- Code with low coverage to be considered unsafe
- Having 100% code coverage not realistic
- 100% code coverage !== working code
- **Low or no** code coverage is the main interest

Enabling code coverage

```
Harness.configure({

    enableCodeCoverage      : true,

    preload : [
        '/ext-all-debug.js',
        '/resources/css/ext-all.css',

        {
            url             : '../gnt-all-debug.js',
            instrument      : true
        }
    ]
});
```

Extra HTML includes

```
<!-- Additional ExtJS file, for charts used by code coverage module -->
<script src="extjs-6.6.0/build/packages/ext-charts/build/ext-charts-debug.js"
type="text/javascript"></script>


<!-- Siesta application -->
<script type="text/javascript" src="../siesta-all.js"></script>

<!-- Additional Siesta files for code coverage feature -->
<script type="text/javascript" src="../siesta-coverage-all.js"></script>
```

Demo code coverage

bin/nodejs examples/nodejs/ --nyc.reporter=html

Advanced features

Extending Siesta

- - Both the `Siesta.Project` and `Siesta.Test` classes are observable so you can be notified when a test starts/finishes
- The Siesta UI is built with Ext JS and can also be overridden just like any other Ext JS application
- `setup` / `tearDown` / `earlySetup` on the Test class

setup / tearDown hooks

- The `setup` hook is called before the start of each test.
- It can be used to perform an async operation (delaying the start of the test)
- This is useful to reset a database for example using a simple Ajax request.
- Similarly, the async `tearDown` test method can be used to do any test finalisation after the test is done.

setup / tearDown

```
Class('My.Test.Class', {

    isa : Siesta.Test.Browser,

    override : {

        setup : function (callback, errback) {
            Ext.Ajax.request({
                url : 'do_login.php',

                success : callback,
                failure : function () {
                    errback('Login failed')
                }
            })
        }
    }
})
```

Overriding the console

```
Project.on('teststart', function (ev, test) {
    var console = test.global.console;

    if (console) {
        console.log =
        console.error =
        console.warn = function () {
            test.fail([].join.apply(arguments));
        };
    }
})
```

Siesta advanced

- `t.chain` accepts a null step which is ignored, flattens array input

- `Siesta.Project#maxThreads` controls number of concurrently running tests (default: 4)

- Project can be configured to break on fail, for rapid debugging. Injects a `debugger;` statement.

Tips n' tricks

- Highly recommended to launch your tests as part of your CI.
- Utilize IDE-integration for rapid test execution
- Don't always simulate just because you can.
  - "`type`" vs `setValue` action

Sencha Cmd app tests

For unit tests of Cmd application code:
Add a global variable setting that prevents your Cmd app from launching

```
Project.configure({
    preload : [
        { text : "window.ISTEST = true;" },
        "app.js" // Built with Cmd, contains all Ext + app JS
    ]
});
```

Ext application configuration

```
Ext.application({
    name: 'ExecDashboard',

    autoCreateViewport: window.ISTEST ? false : 'ExecDashboard.view.main.Main',

    requires: [
        'ExecDashboard.*'
    ]
});
```

Tips and tricks

Testing different screen sizes

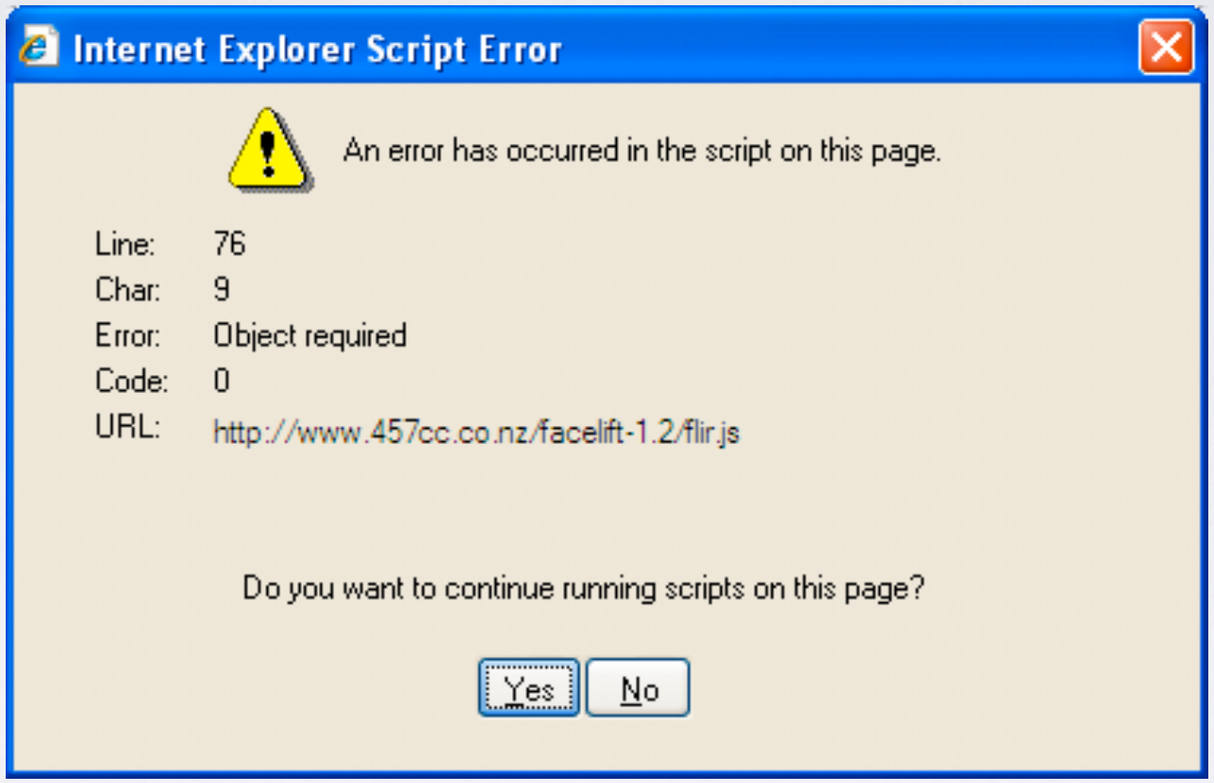
```
{
    name           : 'Responsive app test - landscape',
    viewportWidth  : 1024,
    viewportHeight : 768,
    pageUrl        : 'executive-dashboard/',
    url            : 'executive-dashboard/tests/large-size.t.js'
},
{
    name           : 'Responsive app test - portrait',
    viewportWidth  : 500,
    viewportHeight : 700,
    pageUrl        : 'executive-dashboard/',
    url            : 'executive-dashboard/tests/small-size.t.js'
}
```

SENCHACOMMUNITYDAYS 19

Finding bugs faster

Unhandled JS exception:

What does the user see?


Internet Explorer Script Error
An error has occurred in the script on this page.
Line: 76
Char: 9
Error: Object required
Code: 0
URL: http://www.457cc.co.nz/facelift-1.2/flir.js
Do you want to continue running scripts on this page?
Yes
No

Most likely: **nothing**

Logging unhandled errors

```
win.onerror = function (message, file, line, column, errorObj) {

    var error = message;

    // In latest Chrome, FF
    if (errorObj) {
       error += ‘\r\n’ + errorObj.stack;
    }

    new Image().src = ‘//server.com/error.php?' + error;
};
```

SENCHACOMMUNITYDAYS 19

Try RootCause

ROOTCAUSE

PRICING RESOURCES CONTACT BLOG SIGN IN

Find javascript errors faster

Record and replay user sessions to reproduce javascript errors in your website. Try a demo in our online sandbox.

SIGN UP TRY OUR DEMOS

SENCHACOMMUNITYDAYS 19

That's all folks, questions?