Clutch

Version: 0.1

Release Date: 06/06/2008

Table of Contents

Clutch

by John Leach

Clutch is a classic open source tool – born of a desperate itch which I just simply had to scratch. I decided to give Gears a close scrutiny, since it is an interesting project which provides a browser agnostic plugin platform. I pretty soon realised that I'd be needing a robust JavaScript testing system since it is nearly impossible to debug code within a WorkerPool.

Apart from the usual problems of running JavaScript within a browser, I also wanted a small suite of tools, such as JSLint, JavaScript code file joining or concatenation (as used by the Prototype library), and JavaScript code compression. It seemed a reasonable idea to put them all together within a framework which runs inside the browser itself, much as the unit testing code would have to.

So that's how Clutch got started. Although I'm using it for Gears development, it is actually a library agnostic set of tools for any type of browser based JavaScript development, which coincidentally has Gears support too.

Requirements and Architecture

Clutch consists of a set of HTML pages, CSS and JavaScript files, together with a very rudimentary web application. The Clutch web application is written in Ruby, using WEBrick and ERB. The YUICompressor, used to remove comments and whitespace from JavaScript files, is a Java library (`jar` file).

To use Clutch you will need a Ruby installation (I'm using version 1.8.6) and a Java installation (I'm using version 1.6.0). You definitely need Ruby, but you can avoid using Java if you're prepared to give up JavaScript text compression.

The Clutch web application, apart from serving the static text files, also provides a small set of services, listed below:

Service URL	Description
`/run-compressor`	Compresses a JavaScript file.
`/run-fixture`	Provides POST (write) operations for fixture (unit testing data) files.
`/run-join`	Joins or concatenates a series of JavaScript files into a single file.
`/run-jslint`	Creates a JSLint page for a specific JavaScript file.
`/run-test`	Creates a unit test page, and result view page, for a specific JavaScript file.

Clutch also provides a simple unit testing framework, written in JavaScript, which runs within the browser. The unit test results are written to hard disk automatically in JSON format, and can then be viewed by retrieving the written results.

The following sections give further details of the services and unit testing library.

Running the Clutch Web Application

Clutch can only perform it's magic with the Clutch web application running. To start the Clutch web application, open a command prompt in the `/Clutch/Ruby` folder, then type:

```
ruby server.rb
```

The Clutch web application runs as `localhost` on port `80`, which might conflict with other web servers. After issuing the command the prompt should look something like:

```
C:\Clutch\Ruby>ruby server.rb
[2008-06-08 12:00:40] INFO  WEBrick 1.3.1
[2008-06-08 12:00:40] INFO  ruby 1.8.6 (2007-09-24) [i386-mswin32]
[2008-06-08 12:00:40] INFO  ClutchServer#start: pid=3600 port=80
```

That's it, Clutch is up and running. You can stop the web application at any time by pressing `Ctrl-C`.

Changing the Port

If port 80 does not suite your needs, you can change the value by editing the `server.rb` script. Open `server.rb` in your favourite text editor, move to the end of the file, where you should see:

```
...
# Create the server
server = ClutchServer.new(:Port => 80)

# trap signals to invoke the shutdown procedure cleanly
['INT', 'TERM'].each do |signal|
    trap(signal) { server.shutdown }
end

# Start the server
server.start
```

Change the line

```
server = ClutchServer.new(:Port => 80)
```

to the port value that you want, then restart the server.

Running as a Pseudo Domain

In some circumstances, particularly for Gears development, you may want to use an URL such as `http://clutch.syger.it` instead of `http://localhost`. This can be achieved by setting the `hosts` file.

On Windows you'll find the hosts file in `C:\Windows\system32\drivers\etc`, whereas on most Linux systems it is located in `/etc`. Again, using your favourite text editor, open the file and add a line as follows:

```
127.0.0.1       clutch.syger.it
```

Save the file, and with the Clutch server running, open your browser and type the URL `http://clutch.syger.it` which should now present you with the Clutch home page.

Services

The web application provides a small suite of services which aid in the development, testing, and deployment of JavaScript software.

Compressor Service

The compressor service takes a JavaScript file and removes comments and unnecessary whitespace. This can be a destructive operation in that the original JavaScript file can be overwritten, unless the `-o` option is specified – you have been warned.

To invoke the service, send a `POST` request to `/run_compressor`, appending the absolute JavaScript file path, with respect to the `/Clutch` root directory, as part of the URL. Clutch will then compress the JavaScript file. Additional request parameters can be set to modify the behaviour of the compressor, using the [YUICompressor command line options](#).

The service uses the following default values:

Option	Parameter	Default value
`--line-break`	`line-break`	`0`
`--charset`	`charset`	`UTF-8`
`-o`	`output`	The JavaScript source path. I recommend that you change this value.
`--nomunge`	`nomunge`	`true`

Note that `YUICompressor` options which do not require a value are replaced by a parameter value of `true` or `false`.

Compressor Service Example

```
<form action="/run-compressor/clutch/js/display.js"
    enctype="application/x-www-form-urlencoded" method="post">
  <input name="output" type="hidden" value="clutch/js/min-display.js" />
  <input type="submit" value="Compress 'clutch/js/display.js'" />
</form>
```

See also `/Clutch/joins/index.html`.

Fixture Service

The fixture service allows you to write text to hard disk. When sent a `POST`, the service writes the `POST` data to the file specified in the URL, optionally modifying the contents with parameter values specified in the `POST` request, using ERB.

To invoke the service, send a `POST` request to `/run_fixture`, appending the absolute output text file path, with respect to the `/Clutch` root directory, as part of the URL.

If you specify an absolute input text file path, with respect to the `/Clutch` root directory in the `from` parameter, Clutch will read that file, accepting and executing `include` commands (see the Join Service, below), as well as injecting parameters into the constructed text file.

You can use as many parameters as you like, with the following constraints:

- the `from` parameter name is reserved by Clutch,

- parameter names can only appear once in the request,

- within the text file each parameter value can be referenced by typing `<%= param['parameter-name'] %>`, substituting `parameter-name` for the name of the parameter,

- more complicated expressions can be achieved using ruby code snippets, as explained in the [ERB](#) documentation.

Fixture Service Example

ToDo. I'll be needing this shortly, so there will be an example in the next release.

Join Service

The join (or concatenate) service uses ERB to join together a list of text files, producing a single concatenated file. Each file can contain `include` commands which contain relative URLs to other files to be included at the point of the `include` command itself. This process can also be repeated within the included files (nesting).

To invoke the service, send a `POST` request to `/run_join`, specifying the absolute text file path, with respect to the `/Clutch` root directory, as part of the URL. Clutch will then create the joined (or concatenated) file. The `to` request parameter is required to set the destination absolute URL of the resulting joined file.

Join Service Example

```
<form action="/run-join/joins/clutch/all-tests.js"
    enctype="application/x-www-form-urlencoded" method="post">
  <input name="to" type="hidden" value="/tests/clutch/run/all-tests.js" />
  <input type="submit" value="Join 'unit-test'" />
</form>
```

Given the following folder layout:

```
/joins
  /clutch
    all-tests.js
/clutch
  /src
    unit-test.js
    string.js
    /tests
      all-tests.js
      string-tests.js
      unit-tests.js
```

The relative URL to the `/clutch/src` folder, from within `/joins/clutch/all-`

tests.js will be `../../clutch/src`, giving the following `include` command within `/joins/clutch/all-tests.js`:

```
<%= include '../../clutch/src/string.js',
            '../../clutch/src/tests/unit-test.js' %>
```

Similarly, the relative URL to the `/clutch/src/tests` folder, from within `/clutch/src/tests/all-tests.js` will be `./`, giving the following `include` command:

```
<%= include './string-tests.js', './unit-tests.js' %>
```

See also `/Clutch/joins/index.html`.

JSLint Service

The original [lint program](#) analysed C source code for potential (and subtle) malpractices likely to lead to run-time bugs. Modern C compilers now provide sufficient syntactic and semantic checking that `lint` is now rarely required or used.

Fortunately for JavaScript programmers, [Douglas Crockford](#) has built a lint program specifically for JavaScript, in JavaScript, called [JSLint](#). Finding and removing potentially poor code before unit testing is an essential process, at least for me. Unfortunately, cutting and pasting code to the web page can itself be error prone.

Clutch alleviates this problem by creating static HTML pages that read in your JavaScript code, which can then be analysed locally by JSLint. You only need create the static HTML page once for each JavaScript file you wish to analyse.

To invoke the service, send a `POST` request to `/run_jslint`, specifying the absolute JavaScript file path, with respect to the `/Clutch` root directory, as part of the URL. Clutch will then produce a static HTML file under the `/jslint` folder, which automatically loads the JavaScript file into the JSLint page. You may then wish to add a link to this static HTML file in the `/jslint/index.html` page.

JSLint Service Example

```
<form action="/run-jslint/clutch/js/jslint-loader.js"
    enctype="application/x-www-form-urlencoded" method="post">
  <input type="submit" value="Create 'clutch/js/jslint-loader.js' Page" />
</form>

<a href="/jslint/clutch/js/jslint-loader.html">
  Run /clutch/js/jslint-loader.js page
</a>
```

See also `/Clutch/jslint/index.html`.

Test Service

The test service creates a run/view pair of static HTML files for a given JavaScript file. The reasons for using two HTML files is explained in the 'Unit Testing' section below.

It can also provide functionality similar to the join service. It can use ERB to join together a list of JavaScript files, producing a single concatenated JavaScript file, but

only if the `to` parameter is specified.

The service has three required parameters and three optional parameters listed below:

Parameter	Required	Usage
to	false	Specifies the output absolute URL of the joined JavaScript file to be tested. If not defined, the specified JavaScript file is left unchanged.
run-comment	false	A comment to be displayed in the test run page. Usually used where the tests are expected to take a long time to run.
view-comment	false	A comment to be displayed in the test run page. Usually used where failures and/or errors are expected.
run	true	The absolute URL of the created run test HTML page.
view	true	The absolute URL of the created view test results HTML page.
json	true	The absolute URL of the test results JSON file.

To invoke the service, send a `POST` request to `/run_test`, specifying the absolute JavaScript file path, with respect to the `/Clutch` root directory, as part of the URL. Clutch will then create the two static HTML files.

Test Service Example

The first example joins the specified JavaScript file:

```
<form action="/run-test/joins/clutch/all-tests.js"
    enctype="application/x-www-form-urlencoded" method="post">
  <input name="to" type="hidden" value="/tests/clutch/all-tests.js" />
  <input name="comment" type="hidden"
    value="There should be 1 Failure and 1 Error in these tests,
          &lt;code&gt;testFail&lt;/code&gt; produces the failure, and
          &lt;code&gt;testError&lt;/code&gt; produces the error." />
  <input name="run" type="hidden" value="/tests/clutch/run-all-tests.html" />
  <input name="view" type="hidden" value="/tests/clutch/view-all-tests.html" />
  <input name="json" type="hidden" value="/tests/clutch/all-tests.json" />
  <input type="submit" value="Create Test Pages '/clutch/all-tests'" />
</form>
then <a href="/tests/clutch/run-all-tests.html">run</a>
and <a href="/tests/clutch/view-all-tests.html">view</a> 'all-tests'
```

The second example uses the specified JavaScript file 'as is':

```
<form action="/run-test/tests/clutch/all-tests.js"
    enctype="application/x-www-form-urlencoded" method="post">
  <input name="comment" type="hidden"
    value="There should be 1 Failure and 1 Error in these tests,
          &lt;code&gt;testFail&lt;/code&gt; produces the failure, and
          &lt;code&gt;testError&lt;/code&gt; produces the error." />
  <input name="run" type="hidden" value="/tests/clutch/run-all-tests.html" />
  <input name="view" type="hidden" value="/tests/clutch/view-all-tests.html" />
```

```
  <input name="json" type="hidden" value="/tests/clutch/all-tests.json" />
  <input type="submit" value="Create Test Pages '/clutch/all-tests'" />
</form>
then <a href="/tests/clutch/run-all-tests.html">run</a>
and <a href="/tests/clutch/view-all-tests.html">view</a> 'all-tests'
```

See also `/Clutch/tests/index.html`.

Unit Testing

Unit testing is another useful technique to better ensure the quality and correctness of your JavaScript code. Unfortunately, the dynamic nature of JavaScript makes it a difficult environment in which to perform unit testing. One of the most important aspects is to provide a simple and unintrusive unit test library, which does not alter the characteristics of your own code.

To achieve this objective, Clutch uses a two pass technique. The first pass runs the unit testing code, and stores the results to a JSON file. The second pass reads and then displays the JSON file.

Note that the Clutch unit test framework, due to it's architecture, is not suited for user interface testing, for such needs you might want to use something like Selenium.

In the first pass, Clutch necessarily adds four files to your unit test code:

- `/clutch/src/unit-test.js` – the unit testing framework

- `/clutch/js/json2.js` – the JSON converter

- `/clutch/js/xhr.js` – the XMLHttpResponse function

- `/clutch/js/browser-saver.js` – the JSON file saving function.

Although that may seem like a lot of code, it is kept in two namespaces, `JSON` and `clutch`, in order not to interfere with your own code.

In order for the unit testing process to work, you must supply a `runClutchTests` function in your own code, which either returns a `clutch.unittest` or a `clutch.unittests` object.

Here is an example of a `runClutchTests` function which returns a `clutch.unittest` object:

```
function createUnitTests() {
  return clutch.unittest('Assertion Tests', {

    testPass: function () {
      // ...
    },

    // other tests here

  });
}

function runClutchTests() {
  return createUnitTests();
}
```

Here is an example of a `runClutchTests` function which returns a

```
clutch.unittests object:
  function createUnitTests() {
    return clutch.unittest('Assertion Tests', {

      testPass: function () {
        // ...
      },

      // other tests here

    });
  }

  function createStringTests() {
    return clutch.unittest('String Tests', {

      testTrim: function () {
        // ...
      },

      // other tests here

    });
  }

  function runClutchTests() {
    return clutch.unittests([

      createUnitTests(),

      createStringTests()

    ]);
  }
```

The second pass is independent of your unit testing code, and so can use Prototype to dynamically produce the unit test results display.

See also `/Clutch/clutch/src/tests/` for example unit test code.

The Unit Test Framework

The framework follows a similar pattern to the well known <u>JUnit</u> Java testing framework.

Create your test methods in a plain JavaScript object, then wrap that object in a `clutch.unittest` function call, as shown in the first example above. All functions in your test object which begin with `test` will be executed by the unit test framework, but the order of function execution is not guaranteed.

Before a test*xxx* function is executed, the unit test framework will execute a `setUp` function in your object. After a test*xxx* function has been executed, the unit test framework will execute a `tearDown` function in your object. Clutch provides a default no-operation function for `setUp` and `tearDown` if none are defined in your object.

You can run more than one unit test object by wrapping each in a `clutch.unittests` function call, after you've wrapped the test object in a `clutch.unittest` function call, as shown in the second example above.

When your test object is being executed, the following functions are available:

Function	Purpose
pass(message)	Adds a 'pass' message to the unit test results. Essentially works as a logging function.
fail(message)	Adds a 'fail' message to the unit test results. Use to check that certain code statements are not executed, such as when an exception should have been thrown.
error(/* Error */ err)	Adds an 'error' message to the unit test results. The err parameter should be an instance of Error.
assert(condition, message)	Checks that the expression defined in condition evaluates to true. If not, adds a 'fail' message to the unit test results. The message parameter is optional.

Certain functions are reserved by the unit test framework, and should neither be used nor changed:

Function	Purpose
run(funcName)	Runs a unit test for the specified function name.
summary()	Returns the unit test results.

See also /Clutch/clutch/src/tests/ for example unit test code.

Asynchronous Unit Testing

ToDo. Tricky, but can be done – in the next release.

Using Clutch

With the Clutch web application running, open your browser and type the pseudo domain root address http://clutch.syger.it/, you should then see the Clutch home page.

The Home Page

Home JSLint Join/Compress Unit Tests Documentation

Clutch JSLint Checks

Uses JSLint to check the correctness of the JavaScript code files.

- JSLint Page Creation and Tests

Clutch Join/Compress Utilities

Join together (concatenate) several JavaScript or text files to produce one single JavaScript file.

Compress a (possibly joined and hopefully tested) JavaScript file to produce the smallest possible file.

- Join/Compress File Creation

Clutch Tests

Create static HTML pages, then run and view unit or functional tests on your JavaScript code.

The results are stored in JSON files.

- Unit Test Page Creation

Home JSLint Join/Compress Unit Tests Documentation

The home page provides acces to the principal static HTML pages. You can add your own links as desired for your own projects.

The JSLint Page

Clutch JSLint Page Creation and Tests

JSLint Infrastructure

- Create 'clutch/js/xhr.js' Page | then Run /clutch/js/xhr.js page
- Create 'clutch/js/jslint-loader.js' Page | then Run /clutch/js/jslint-loader.js page

Unit Testing Infrastructure

- Create 'clutch/js/json2.js' Page | then Run /clutch/js/json2.js page
- Create 'clutch/js/browser-saver.js' Page | then Run /clutch/js/browser-saver.js page
- Create 'clutch/js/display.js' Page | then Run /clutch/js/display.js page

Clutch Library

- Create 'clutch/src/browser.js' Page | then Run /clutch/src/browser.js page
- Create 'clutch/src/string.js' Page | then Run /clutch/src/string.js page
- Create 'clutch/src/unit-test.js' Page | then Run /clutch/src/unit-test.js page

Clutch Gears Library

- Create 'clutch/src/gears/xhr.js' Page | then Run /clutch/src/gears/xhr.js page

The main JSLint page provides links both for the creation of static JSLint pages, and for running JSLint on the JavaScript code. Each page loads the latest version of the JavaScript code, and so can be used to check modifications 'on the fly'. You may need to refresh the page (usually F5).

JSLint Results

JSLint Test: /clutch/src/string.js page

The generation of a static JSLint HTML page gives a results page which also contains a link to the newly created page.

JSLint: The JavaScript Verifier:
'/clutch/src/string.js'

Options, Documentation, Book

Edition 2008-06-04

```
/*
 */

/*jslint evil: true */
/*global clutch, JSON */

if (!this.clutch) {
    clutch = {};
}

clutch.date = {
    toStandardJSON: function () {
        function tens(n) {
            // Format integers to have at least two digits.
            return n < 10 ? '0' + n : n;
        }
```

JSLint

Options

☐ Stop on first error
☐ Strict whitespace
☑ Assume a browser
☐ Assume a Yahoo Widget
☐ Assume a Windows Sidebar Gadget
☐ Assume Rhino
☐ ADsafe

☑ Disallow undefined variables
☑ Disallow leading _ in identifiers
☑ Disallow == and !=
☑ Disallow ++ and --
☑ Disallow bitwise operators
☐ Disallow . in RegExp literals
☑ Disallow global var

☐ Tolerate debugger statements
☐ Tolerate eval
☐ Tolerate HTML case
☐ Tolerate HTML event handlers
☐ Tolerate HTML fragments
☐ Tolerate sloppy line breaking
☐ Tolerate unfiltered for in

Clear All Options   Recommended Options   Good Parts

Indentation 4   Predefined (, separated)

Copyright 2002 Douglas Crockford. All Rights Reserved Wrrrldwide and Beyond!
Code Conventions for the JavaScript Programming Language.
Join the JSLint Group.

The JSLint static HTML page when first loaded. Click on the 'JSLint' button to see the results.

JSLint: The JavaScript Verifier:
'/clutch/src/string.js'

JSLint

Options, Documentation, Book

Edition 2008-06-04

```
/*

*/

/*jslint evil: true */
/*global clutch, JSON */

if (!this.clutch) {
    clutch = {};
}

clutch.date = {
    toStandardJSON: function () {
        function tens(n) {
            // Format integers to have at least two digits.
            return n < 10 ? '0' + n : n;
        }
```

[JSLint]

No new global variables introduced.

13 "toStandardJSON"()
        Closure hundreds, tens
        Global Date

/*members UTC, arg, clutch, date, endsWith, evalJSON, exec, fromJSON,
    getTime, getUTCDate, getUTCFullYear, getUTCHours, getUTCMilliseconds,
    getUTCMinutes, getUTCMonth, getUTCSeconds, indexOf, inspect, isJSON,
    lastIndexOf, length, match, message, messagePack, messageUnpack, parse,
    prototype, replace, slice, startsWith, string, stringify, substring,
    toClutchJSON, toJSON, toMicrosoftJSON, toStandardJSON, toString, trim,
    unfilterJSON
*/

Options

- ☐ Stop on first error
- ☐ Strict whitespace
- ☑ Assume a browser
- ☐ Assume a Yahoo Widget
- ☐ Assume a Windows Sidebar Gadget
- ☐ Assume Rhino
- ☐ ADsafe
- ☑ Disallow undefined variables
- ☑ Disallow leading _ in identifiers
- ☑ Disallow == and !=
- ☑ Disallow ++ and --
- ☑ Disallow bitwise operators
- ☐ Disallow . in RegExp literals
- ☑ Disallow global var

- ☐ Tolerate debugger statements
- ☐ Tolerate eval
- ☐ Tolerate HTML case
- ☐ Tolerate HTML event handlers
- ☐ Tolerate HTML fragments
- ☐ Tolerate sloppy line breaking
- ☐ Tolerate unfiltered for in

[Clear All Options] [Recommended Options] [Good Parts]

Indentation [4]   Predefined (, separated) [   ]

Copyright 2002 Douglas Crockford. All Rights Reserved Wrrrldwide and Beyond!
Code Conventions for the JavaScript Programming Language.
Join the JSLint Group.

The JSLint static HTML page after the 'JSLint' button has been pressed (note that the results have been truncated to fit on the page). Error messages are shown in a

shocking pink, probably to encourage you to correct them.

The Join/Compress Page

Home JSLint Join/Compress Unit Tests Documentation

Clutch Join/Compress File Creation

JSLint Infrastructure

- Compress '/clutch/js/xhr.js'
- Compress '/clutch/js/jslint-loader.js'
- Compress '/clutch/js/jslint/webjslint.js'

Unit Testing Infrastructure

- Compress '/clutch/js/json2.js'
- Compress '/clutch/js/browser-saver.js'

Unit Testing Infrastructure - Prototype Display

- Compress '/clutch/js/display.js'
- Compress '/clutch/js/prototype/prototype.js'
- Compress '/clutch/js/prototype/builder.js'

Unit Tests

- First: Join 'all-tests' then: Compress 'all-tests'
- First: Join 'string-tests' then: Compress 'string-tests'
- First: Join 'unit-test-tests' then: Compress 'unit-test-tests'

Home JSLint Join/Compress Unit Tests Documentation

The main Join/Compress HTML page provides links for the joining (concatenation) or compression of the principal JavaScript files that make up Clutch.

Home JSLint Join/Compress Unit Tests Documentation

Join Results

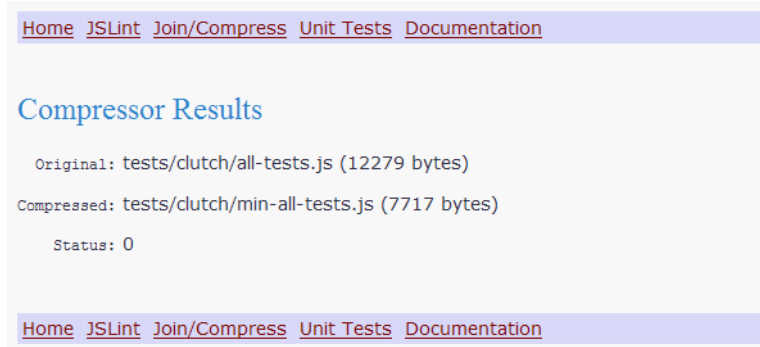From: joins/clutch/all-tests.js
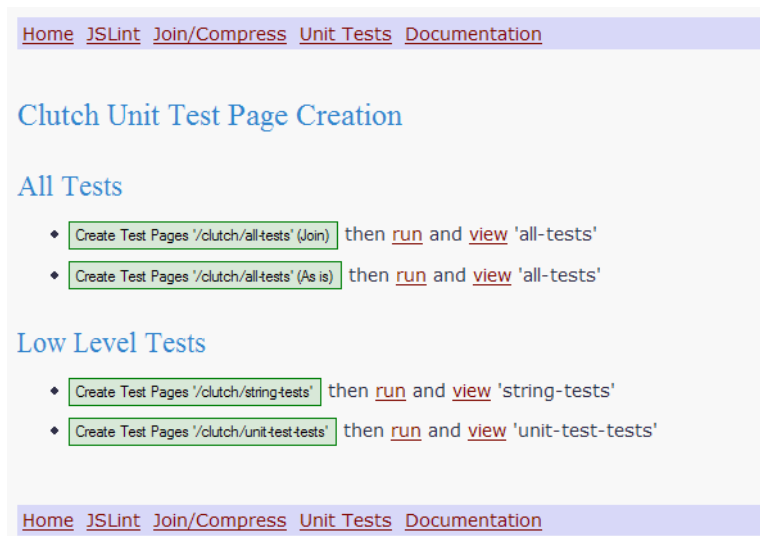
To: tests/clutch/all-tests.js (12279 bytes)

Home JSLint Join/Compress Unit Tests Documentation

Each Join command produces a results page indicating the number of bytes in the final JavaScript file.

Compressor Results

```
   Original: tests/clutch/all-tests.js (12279 bytes)
 Compressed: tests/clutch/min-all-tests.js (7717 bytes)
     Status: 0
```
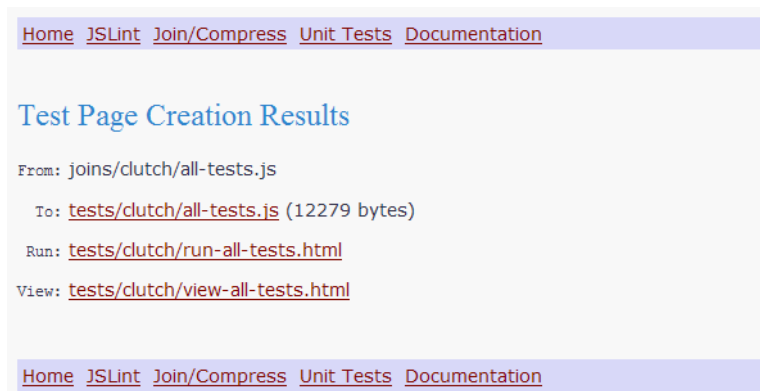
Each Compress command produces a results page indicating both the original JavaScript file byte size, and the compressed JavaScript file byte size.

The Unit Tests Page

Clutch Unit Test Page Creation

All Tests

- Create Test Pages '/clutch/all-tests' (Join) then run and view 'all-tests'
- Create Test Pages '/clutch/all-tests' (As is) then run and view 'all-tests'

Low Level Tests

- Create Test Pages '/clutch/string-tests' then run and view 'string-tests'
- Create Test Pages '/clutch/unit-test-tests' then run and view 'unit-test-tests'

The main Unit Test page provides links both for the creation of the static HTML test pages, and for running or viewing the results.

Test Page Creation Results

```
From: joins/clutch/all-tests.js
  To: tests/clutch/all-tests.js (12279 bytes)
 Run: tests/clutch/run-all-tests.html
View: tests/clutch/view-all-tests.html
```

Each test page creation command produces a results page with links to the run test page, and the view test results page. In this example the JavaScript file was also joined (concatenated).

Test Page Creation Results

Code: tests/clutch/all-tests.js

Run: tests/clutch/run-all-tests.html

View: tests/clutch/view-all-tests.html
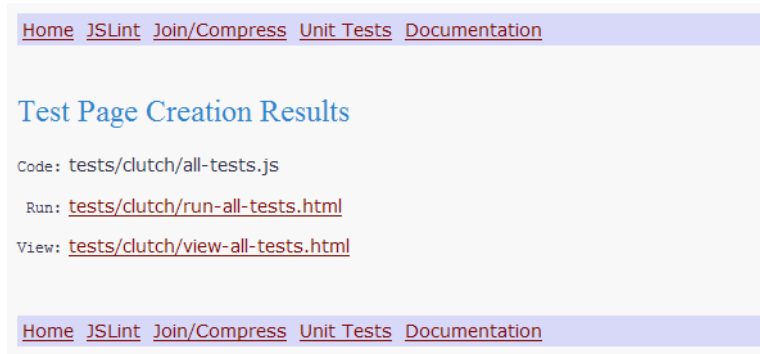
This example uses the JavaScript file 'as-is', again producing run tests and view results links.

Clutch Run Test: 'tests/clutch/all-tests.js'

Unit tests completed and stored.
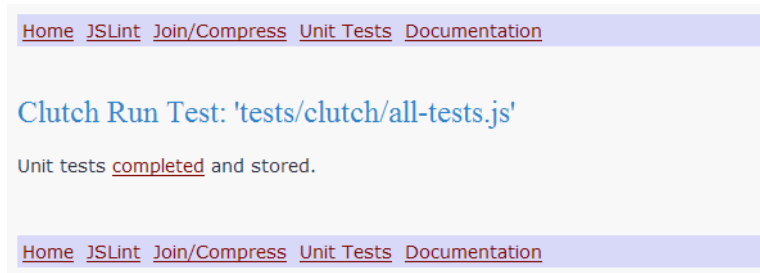
The run test page will start running the unit tests automatically. Once the tests have been completed (and the results stored as a JSON file) the link to the view results page is displayed.

 Clutch

Clutch View Test 'tests/clutch/all-tests.js'

There should be 2 Failures and 1 Error in these tests, `testFail` produces the failures, and `testError` produces the error.

Summary:

Tests	Failures	Errors	Success Rate	Time (ms)
31	2	1	90.32%	0

Unit Tests Summary:

Unit Test Name	Tests	Failures	Errors	Success Rate	Time (ms)
Assertion Tests	6	2	1	50.00%	0
String Tests	25	0	0	100.00%	0

Unit Test: Assertion Tests

Function	Tests	Failures	Errors	Time (ms)
testPass	2	0	0	0
testFail	2	2	0	0
testError	1	0	1	0
testAssert	1	0	0	0

Unit Test: String Tests

Function	Tests	Failures	Errors	Time (ms)
testTrim	2	0	0	0
testStartsWith	4	0	0	0
testEndsWith	4	0	0	0
testJsonObject	4	0	0	0
testJsonArray	4	0	0	0
testMessagePack	7	0	0	0

All errors:

Unit Test Name	Function	Reason
Assertion Tests	testError	http://clutch.syger.it/tests/clutch/all-tests.js(314) Error: Test Error("Test error() call")@:0 ()@http://clutch.syger.it/tests/clutch/all-tests.js:314 ("testError")@http://clutch.syger.it/clutch/src/unit-test.js:67 ()@http://clutch.syger.it/clutch/src/unit-test.js:149 ()@http://clutch.syger.it/clutch/src/unit-test.js:176 storeClutchTests(runClutchTests,"/run-fixture/tests/clutch/all-tes onload([object Event])@http://clutch.syger.it/tests/clutch/run-all @:0

All failures:

Unit Test Name	Function	Reason
Assertion Tests	testFail	Test fail() call
		assert(false) guaranteed to fail

The unit test results page displays the results (read from the intermediary JSON file). Errors and failures are displayed in separate lists.
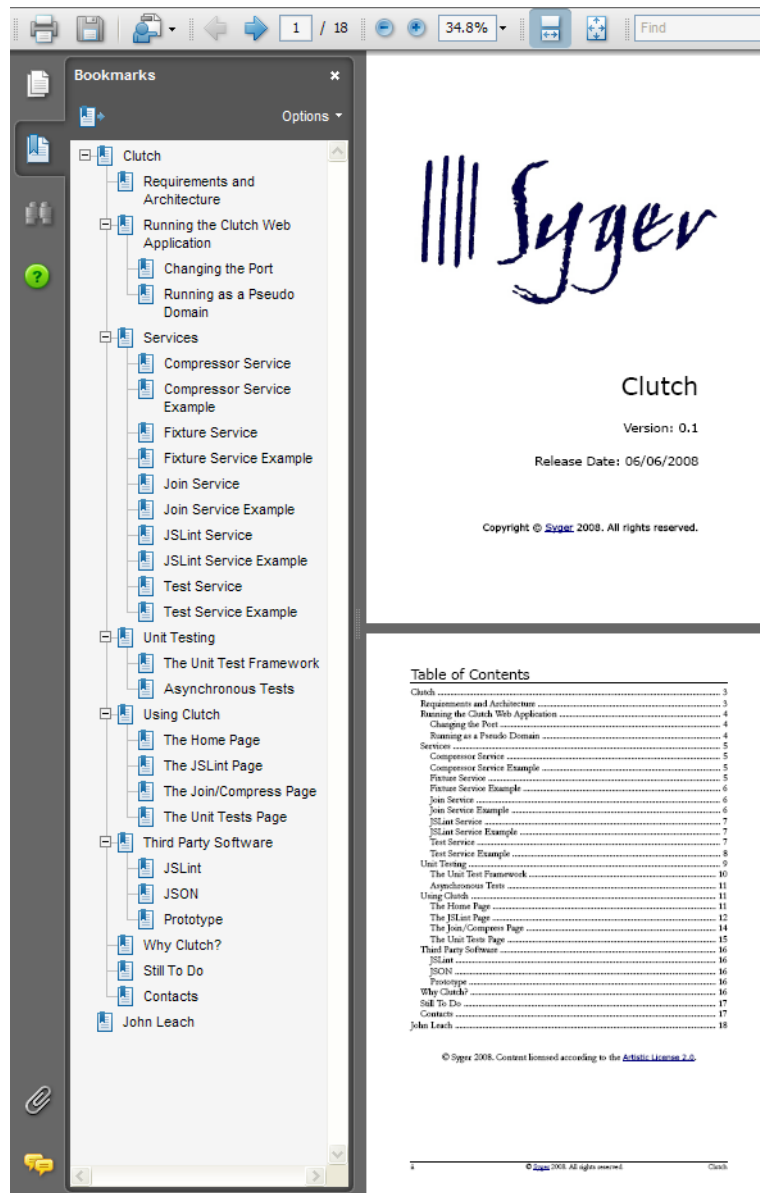
Documentation

This document is also available from within the Clutch web application.

Third Party Software

Clutch stands on the shoulders of giants. There are three important JavaScript files used by Clutch; JSLint, JSON, and Prototype. All three have been slightly modified for one reason or another, described below.

JSLint

To overcome a parsing bug in Opera, the seven regular expressions, `ax`, `cx`, `tx`, `lx`, `ix`, `jx` and `ux` were converted to string syntax format (at about line 475).

JSON

To overcome a parsing bug in Opera, the two regular expressions, `cx` and `escapable` were converted to string syntax format (at about line 180).

In order to run within a WorkerPool, the `eval(text)` call is replaced with a `new Function(text)()` statement (at about line 456).

Prototype

The Prototype library itself has not been modified, but the Clutch string library provides substitutions for `Date.prototype.toJSON()` and `String.prototype.evalJSON()` which allow for the Microsoft Date format, and a Clutch derivative in JSON text.

Why the Name Clutch?

Two reasons, firstly because I felt that I was clutching at straws, and secondly it is the mechanism that lies between the engine - your code - and the gearbox - the browser, or Gears, in my case. It is also the third pedal (the one on the left) in a motor car, which is usually missing on American cars, because they nearly all have automatic gearboxes. I felt it was also the 'missing pedal' in a Gears development environment.

Contacts

Syger can be contacted for consultancy work on any of the topics mentioned in this article, by sending an email to **info@syger.it**.

John Leach

I'm a professional programmer, and Chief Technical Officer of a small software house in Verona, Italy, called Syger. The name came about from being influenced by a drawing by Roger Dean, of ferocious, intelligent badgers, which I transposed to the **S**iberian T**iger**, my favourite animal from childhood, hence Syger.

Most of the work done by my company is consultancy and software development for other software houses.

I now spend most of my time divided between scripting languages and frameworks such as Ruby, Groovy, Ruby on Rails, and Grails, and my old time favourites, Java and JavaScript.