Frizione - Clutch

Version: 0.3

Release Date: 26/06/2008

Table of Contents

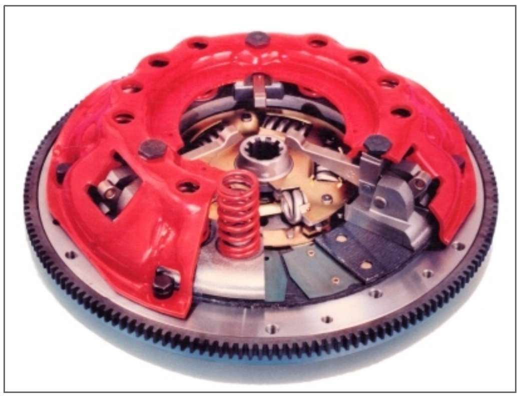
Frizione - Clutch 4
- Requirements and Architecture 4
- Running the Clutch Web Application 5
  - Changing the Port 5
  - Running as a Pseudo Domain 6
- Services 6
  - Compressor Service 6
  - Compressor Service Example 7
  - Fixture Service 7
  - Fixture Service Example 7
  - Join Service 8
  - Join Service Example 8
  - JSLint Service 9
  - JSLint Service Example 9
  - Test Service 9
  - Test Service Example 10
- Unit Testing 11
  - WorkerPool Unit Testing 12
  - The Unit Test Framework 13
  - Asynchronous Unit Testing 14
- Creating Your Own Projects 15
- Using Clutch 16
  - The Home Page 16
  - The JSLint Page 17
  - The Join/Compress Page 20
  - The Unit Tests Page 22
  - Documentation 25
- Third Party Software 25
  - JSLint 25
  - JSON 26
  - Prototype 26
- Why the Name Clutch? 26
- Contacts 26

The Clutch Library 27
- Browser 27
- Introspect 27
- String 27
- Gears 29
- Timer 29
- XHR 30
- WorkerPool Messages 31
- Database Utilities 31
- Database Logger 32

Clutch RoadMap 34
The Good Parts 34
The So-So Parts 34
The Bad Parts 35
The Missing Parts 35
Frizione – Clutch Version History 37
Version 0.3 – 26/06/2008 37
Version 0.2 - 18/06/2008 37
Version 0.1.1 – 10/06/2008 37
Version 0.1 - 10/06/2008 37
John Leach 38

Frizione - Clutch

by John Leach



Frizione (that's Italian for Clutch, which is what I'll call the project from now on) 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 Frizione, er, 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.

The Frizione project is hosted on the Google Code web site. It is released under a MIT license, and kept in a shady Subversion repository, away from direct sunlight. There is also a low volume discussion group.

Requirements and Architecture

Clutch consists of a set of HTML pages, CSS and JavaScript files, together with a very rudimentary, but essential web application. The Clutch web application is written in Ruby, using WEBrick and ERB. The YUICompressor, which is *optionally* 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 *optionally*, 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 settle for JSMin (the Ruby version) JavaScript text compression instead of YUICompressor.

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 `/Frizione/Ruby` directory, 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:\Frizione\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. Since this can be a destructive operation, Clutch checks that you supply an output URL, and that it is not identical to the input URL.

To invoke the service, send a `POST` request to `/run_compressor`, appending the absolute JavaScript file path, with respect to the `/Frizione` root directory, as part of the URL. Clutch will then compress the JavaScript file, either using JSMin or YUICompressor. 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:

Parameter	Option	Required	Default value
`jsmin`	No equivalent	`no`	`true`. Set to `false` to use the YUICompressor.
`line-break`	`--line-break`		`0`
`charset`	`--charset`		`UTF-8`
`output`	`-o`	`yes`	None.
`nomunge`	`--nomunge`		`true`

Note that `YUICompressor` options which do not require a value (such as

`--nomunge`) are replaced by a parameter value of `true` or `false`.

Note also that JSMin ignores all parameters except `jsmin` and `output`.

Compressor Service Example

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

The first example above, uses JSMin to compress the JavaScript file.

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

This second example uses YUICompressor to compress the JavaScript file.

See also `/Frizione/projects/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 `/Frizione` root directory, as part of the URL.

If you specify an absolute input text file path, in the `from` parameter, with respect to the `/Frizione` root directory, 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, and any parameter names starting with `clutch`, are reserved by Clutch,
- within the text file each parameter value can be referenced by typing `<%= params['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

The fixture service is used by the run unit test HTML pages to store the JSON formatted test results file. A JavaScript example of this usage is shown below:

```
function storeClutchTests(testFunction, jsonUrl, viewUrl) {

  jsonUrl = '/run-fixture' + jsonUrl;
  ...
  clutch.executeRequest("POST", jsonUrl, null,
      JSON.stringify(tests.summarise(), null, "\t"), handleRequest);
}
```

Here `jsonUrl` is the file path where the results are stored, and the POST body is created by the `JSON` library `stringify` function.

See `/Frizione/clutch/js/saver.js` for the complete code example.

TODO: show true fixtures examples...

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

Given the following directory layout:

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

The relative URL to the `/projects/clutch/src` directory, from within `/projects/clutch/joins/all-tests.js` will be `../src`, giving the following `include` command within `/projects/clutch/joins/all-tests.js`:

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

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

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

See also `/Frizione/projects/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 `/Frizione` root directory, as part of the URL. Clutch will then produce a static HTML file specified by the `to` parameter, which automatically loads the JavaScript file ready for linting.

JSLint Service Example

```
<form action="/run-jslint/projects/clutch/src/introspect.js"
    enctype="application/x-www-form-urlencoded" method="post">
  <input name="to" type="hidden"
      value="/projects/clutch/jslint/src/introspect.html" />
  <input type="submit" value="Create 'clutch/src/introspect.js' Page" />
</form>
```

```
<a href="/projects/clutch/jslint/src/introspect.html">
  Run /clutch/src/introspect.js page
</a>
```

See also `/Frizione/projects/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 four optional parameters listed below:

Parameter	Required	Usage
`to`	no	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`	no	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`	no	A comment to be displayed in the test run page. Usually used where failures and/or errors are expected.
`gears`	no	When set to "`gears`" adds Gears initialisation code to the test page. When set to "`workerpool`" adds `WorkerPool` unit testing code to the test page.
`run`	yes	The absolute URL of the created run test HTML page.
`view`	yes	The absolute URL of the created view test results HTML page.
`json`	yes	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 `/Frizione` root directory, as part of the URL. Clutch will then create the two static HTML files.

Test Service Example

The first example joins, and then tests, the specified JavaScript file:

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

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

```
<form action="/run-test/projects/clutch/tests/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;failTest&lt;/code&gt; produces the failure, and
             &lt;code&gt;errorTest&lt;/code&gt; produces the error." />
  <input name="run" type="hidden"
      value="/projects/clutch/tests/run-all-tests.html" />
  <input name="view" type="hidden"
      value="/projects/clutch/tests/view-all-tests.html" />
  <input name="json" type="hidden"
      value="/projects/clutch/tests/all-tests.json" />
  <input type="submit" value="Create Test Pages '/clutch/all-tests'" />
</form>
then <a href="/projects/clutch/tests/run-all-tests.html">run</a>
and <a href="/projects/clutch/tests/view-all-tests.html">view</a> 'all-tests'
```

See also `/Frizione/projects/clutch/tests/index.html`. `WorkerPool` unit testing examples can be found in `/Frizione/projects/clutch-gears/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 consider something like Selenium.

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

- `/projects/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/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`, so as 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.test.unit` or a `clutch.test.group` object.

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

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

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

      // other tests here

    }, 1000);
  }

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

The `clutch.test.unit` function requires three parameters; the name of the unit test, the object to test, and a maximum timeout period in milliseconds.

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

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

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

      // other tests here

    }, 1000);
  }

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

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

      // other tests here

    }, 1000);
  }

  function runClutchTests() {
    return clutch.test.group([
      createUnitTests(),
      createStringTests()
    ], 2000);
  }
```

The `clutch.test.group` function requires two parameters; an array of `clutch.test.unit` objects to test, and a maximum timeout period in milliseconds.

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 `/Frizione/projects/clutch/src/tests/` and `/Frizione/projects/clutch-gears/src/tests/` for example unit test code.

WorkerPool Unit Testing

Clutch also provides extensions to the unit testing framework to allow you to test your code within a `WorkerPool` environment. In this case, additional JavaScript files must be added to your code which will then be loaded together into a `WorkerPool` instance. These files are:

- `/projects/clutch/src/unit-test.js` – the unit testing framework,
- `/projects/clutch/src/gears/wp-messages.js` – the `WorkerPool` message handling functions,
- `/projects/clutch/src/gears/wp-unit-test.js` – the `WorkerPool` Unit Test function.

Although that may also seem like a lot of code, it is kept in one namespace, `clutch`, so as not to interfere with your own code.

The Unit Test Framework

The framework follows a similar pattern to the well known JUnit Java testing framework.

Create your test methods in a plain JavaScript object, then wrap that object in a `clutch.test.unit` 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.test.group` function call, after you've wrapped each test object in a `clutch.test.unit` function call, as shown in the second example above.

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

Function	Purpose
`this.log(message)`	Adds a 'log' `message` to the unit test results. Essentially works as a logging function, where traditional `console` functions are not available.
`this.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.
`this.error(/* Error */ err)`	Adds an 'error' message to the unit test results. The `err` parameter should be an instance of `Error`. Not usually required of the unit test code, as all errors are caught and logged by Clutch.
`this.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.

See also `/Frizione/projects/clutch/src/tests/` for example unit test code. Additionally, `/Frizione/projects/clutch-gears/src/tests/gears/` contains Gears specific unit test code.

When the usual `test`*xxx* function naming convention is not suitable, or when you need to control the order of test function calls, Clutch provides a meta-programming mechanism of specifying the test methods, described below.

Asynchronous Unit Testing

Clutch can also perform asynchronous unit testing, but needs a little help from you, the programmer. Each asynchronous test must consist of a synchronous function, and zero or more asynchronous functions which you expect the system under test to call. Clutch only checks the first asynchronous function called, it currently has no provision for checking multiple asynchronous function calls triggered by a single synchronous function call.

The help that Clutch requires from you, is in the form of a small JSON like property within your test object with the name `clutchTest`. The following example shows the meta-programming information:

```
function createXhrTests() {

  return clutch.test.unit('XHR Tests', {

    clutchTests: [
      { func: 'validUrl', callbacks: [ 'validUrlHandler' ] },
      { func: 'invalidUrl', callbacks: [ 'invalidUrlHandler' ] },
      { func: 'abortedRequest', callbacks: [ 'abortedRequestHandler' ] }
    ],
    validUrl: function () {
      // ...
    },

    validUrlHandler: function (status, statusText, responseText) {
      // ...
    },

    // other tests here

  }, 18000);
}
```

The `clutchTests` property consists of an array of objects, each of which contain two properties; `func`, the name of the synchronous function, and `callbacks`, an array of callback functions, or `null` for a pure synchronous test.

The `clutchTests` property can also be used to guarantee the order of a set of synchronous unit tests, or to create a mix of synchronous and asynchronous tests, which again will be run in the specified order. If Clutch finds the `clutchTests` property in your test object, it will not look for the traditional `test`*xxx* functions.

In the following example a set of synchronous tests are executed in the specified order:

```
  function createUnitTests() {

    return clutch.test.unit('Assertion Tests', {

      clutchTests: [
        { func: 'logTest', callbacks: null },
        { func: 'passTest', callbacks: null },
        { func: 'failTest', callbacks: null },
        { func: 'errorTest', callbacks: null },
        { func: 'assertTest', callbacks: null }
      ],

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

      // other tests here

    }, 1000);
  }
```

See also `/Frizione/projects/clutch/src/tests/unit-test-tests.js` for an example of synchronous unit test code, and `/Frizione/projects/clutch/src/tests/gears/xhr-tests.js` for example asynchronous unit test code.

Creating Your Own Projects

Clutch lives a quiet life in a shady Subversion repository. Unfortunately this can make adding your own code to the Clutch framework difficult and potentially dangerous. However, from version 0.2 onwards, Clutch provides a `/projects` directory in which you can safely store your own JavaScript code, and keep it under the loving care of your own Subversion repository, while still being able to update both your own code, and Clutch itself. The Clutch library is also stored in the `/projects/clutch` directory. Whether this can be considered a form of bootstrapping or dog food consumption is a matter of opinion.

Connecting your project to Clutch is relatively simple:

- add a directory for your project under `/projects`,
- create a `clutch.json` file in that folder.

The `clutch.json` file contains the project name, and links to your project's principal directories. The following is the clutch project `clutch.json` file:

```
{
  "name":   "Frizione - Clutch",
  "home":   "/projects/clutch/",
  "joins":  "/projects/clutch/joins/",
  "jslint": "/projects/clutch/jslint/",
  "tests":  "/projects/clutch/tests/"
}
```

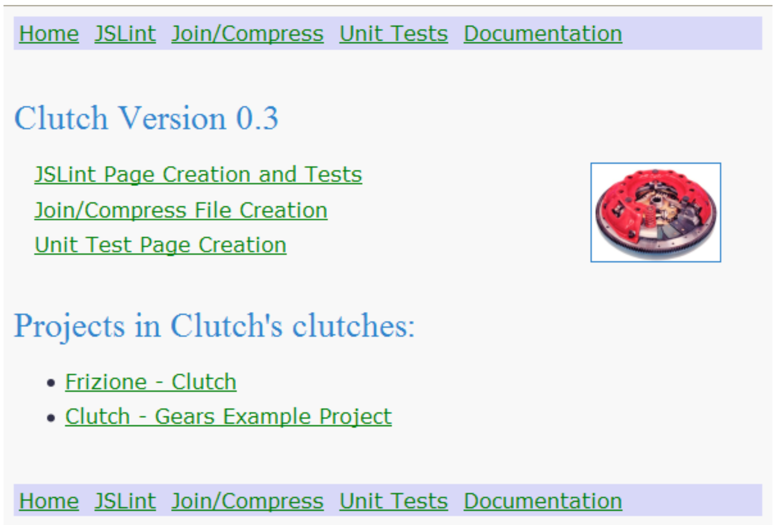
Only the `name` and `home` values are required. Once you have added this file, you should see your project name listed under “Projects in Clutch's clutches” in the home page (`http://clutch.syger.it/`).

See `/Frizione/projects/clutch` and `/Frizione/projects/clutch-gears` for examples of creating an external project.

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 Version 0.3

JSLint Page Creation and Tests
Join/Compress File Creation
Unit Test Page Creation

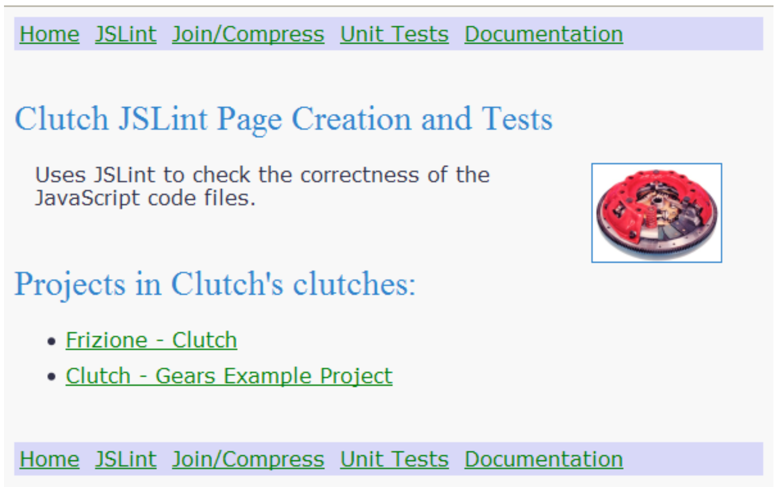
Projects in Clutch's clutches:

- Frizione - Clutch
- Clutch - Gears Example Project

Home JSLint Join/Compress Unit Tests Documentation

The home page provides access to the principal static HTML pages, and the project pages.

The JSLint Page


Home JSLint Join/Compress Unit Tests Documentation

Clutch JSLint Page Creation and Tests

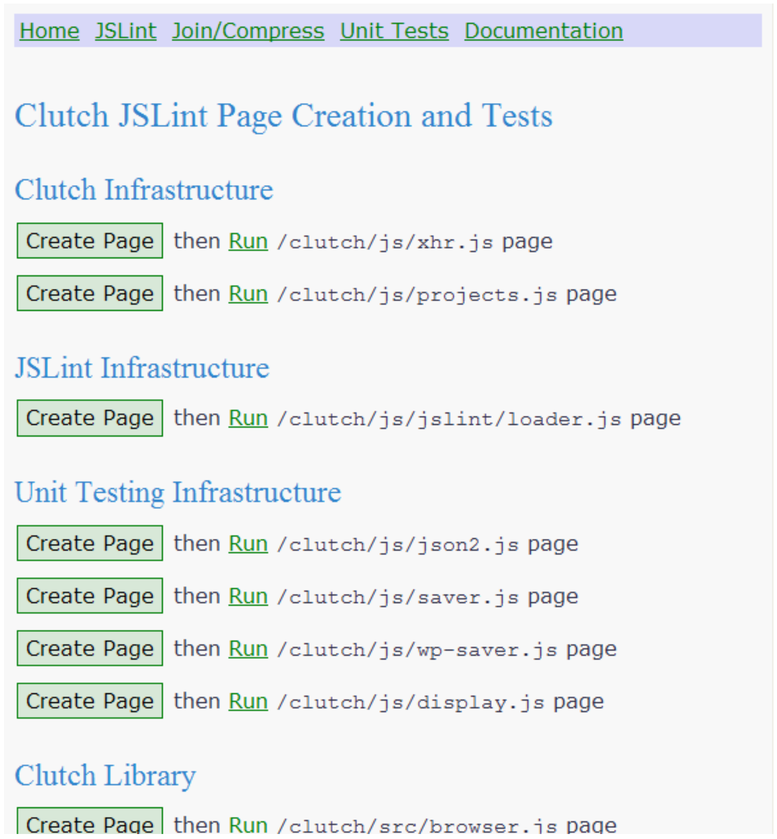
Uses JSLint to check the correctness of the JavaScript code files.

Projects in Clutch's clutches:

- Frizione - Clutch
- Clutch - Gears Example Project

Home JSLint Join/Compress Unit Tests Documentation

The main JSLint page provides links to each project's JSLint page (if specified in the project's `clutch.json` file). Each individual project page contains links both for the creation of static JSLint pages, and for running JSLint on the JavaScript code.


Home JSLint Join/Compress Unit Tests Documentation

Clutch JSLint Page Creation and Tests

Clutch Infrastructure

Create Page then Run /clutch/js/xhr.js page

Create Page then Run /clutch/js/projects.js page

JSLint Infrastructure

Create Page then Run /clutch/js/jslint/loader.js page

Unit Testing Infrastructure

Create Page then Run /clutch/js/json2.js page

Create Page then Run /clutch/js/saver.js page

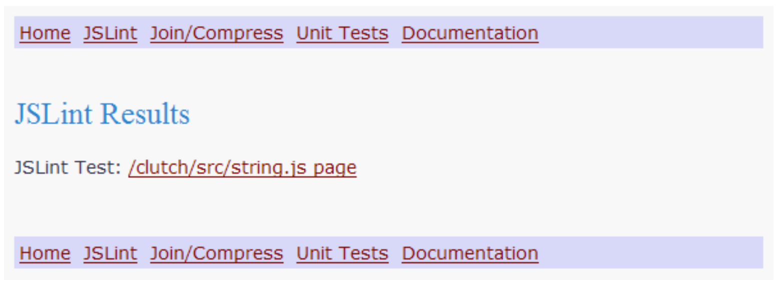
Create Page then Run /clutch/js/wp-saver.js page

Create Page then Run /clutch/js/display.js page

Clutch Library

Create Page then Run /clutch/src/browser.js page

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`).

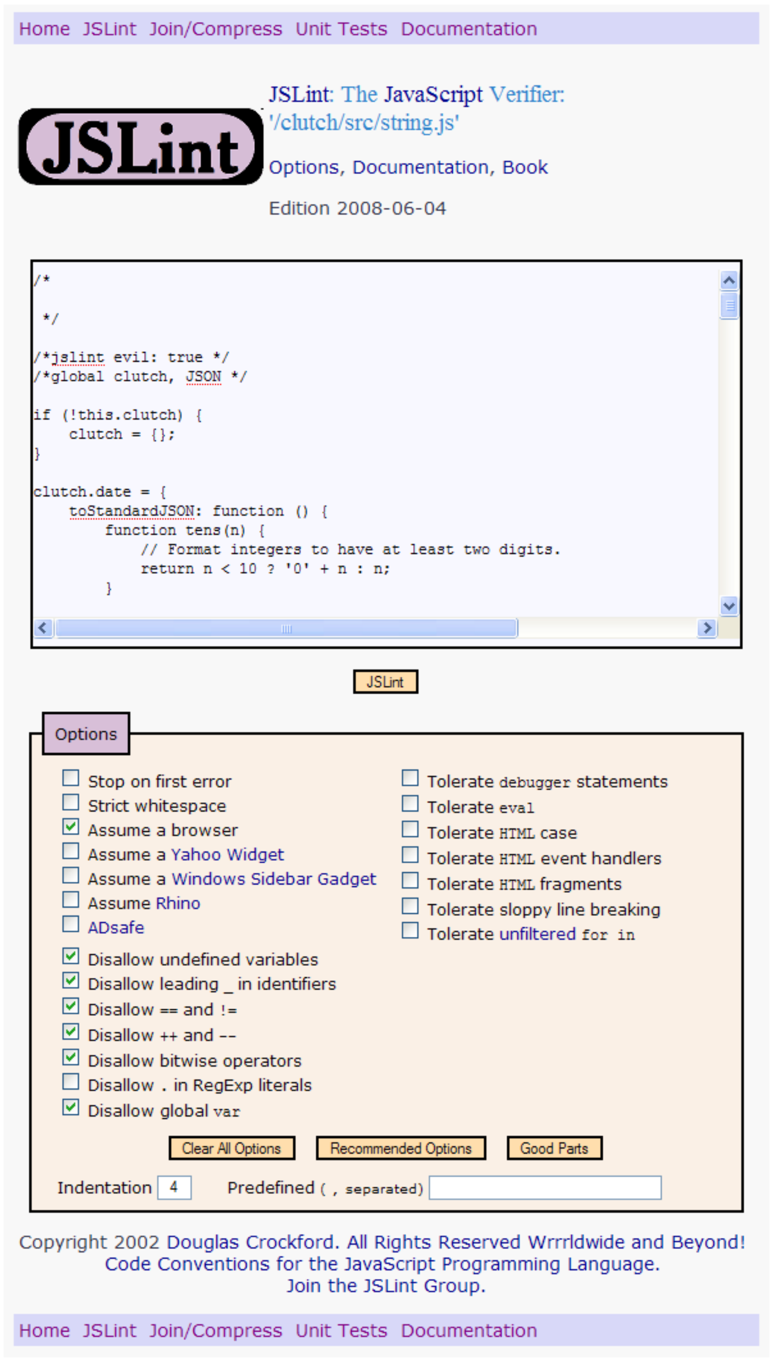

Home JSLint Join/Compress Unit Tests Documentation

JSLint Results

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

Home JSLint Join/Compress Unit Tests Documentation

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


Home JSLint Join/Compress Unit Tests Documentation

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

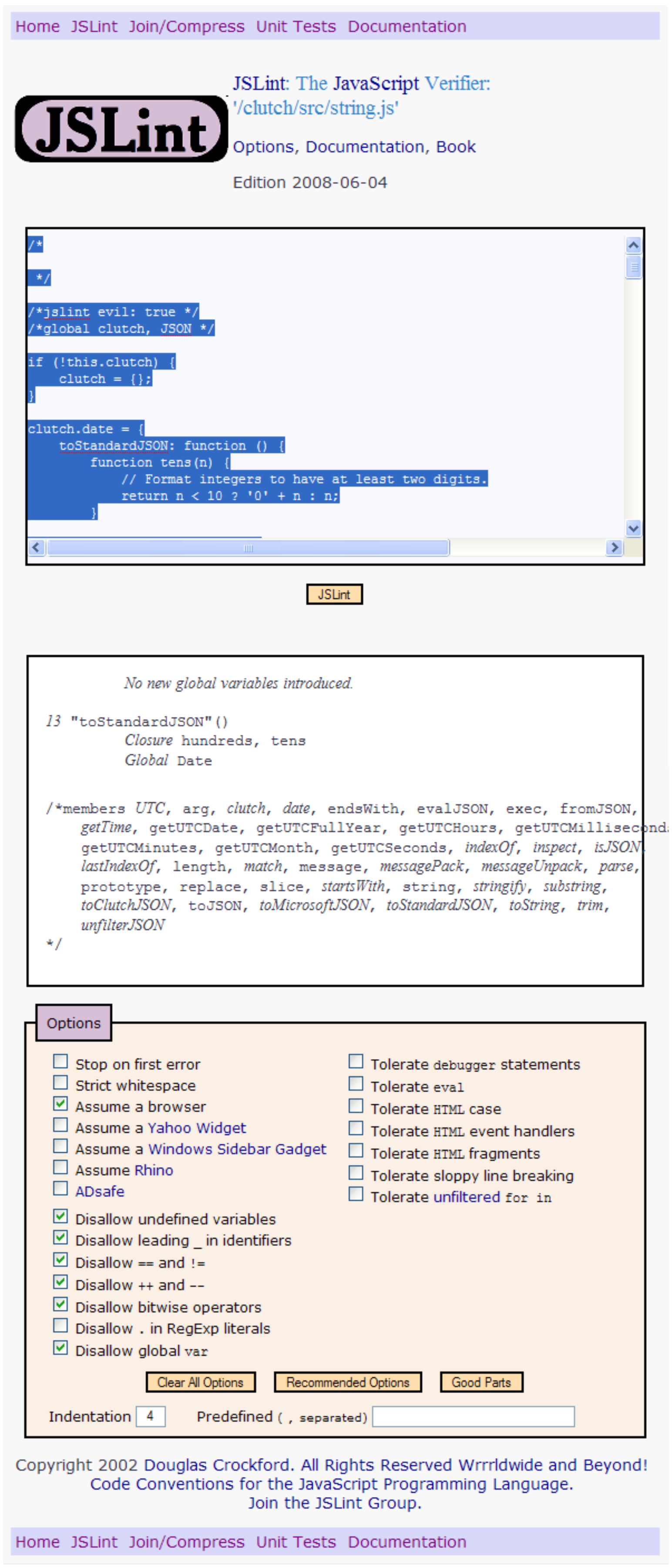
Clear All Options | Recommended Options | Good Parts

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.

Home JSLint Join/Compress Unit Tests Documentation

The JSLint static HTML page when first loaded. Click on the 'JSLint' button to see the results. Refresh the page if you change your JavaScript source code.


Home JSLint Join/Compress Unit Tests Documentation

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;
        }
```

```
          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, getUTCMillisecond
    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	☐ Tolerate debugger statements
☐ Strict whitespace	☐ Tolerate eval
☑ Assume a browser	☐ Tolerate HTML case
☐ Assume a Yahoo Widget	☐ Tolerate HTML event handlers
☐ Assume a Windows Sidebar Gadget	☐ Tolerate HTML fragments
☐ Assume Rhino	☐ Tolerate sloppy line breaking
☐ ADsafe	☐ Tolerate unfiltered for in
☑ Disallow undefined variables	
☑ Disallow leading _ in identifiers	
☑ Disallow == and !=	
☑ Disallow ++ and --	
☑ Disallow bitwise operators	
☐ Disallow . in RegExp literals	
☑ Disallow global var	

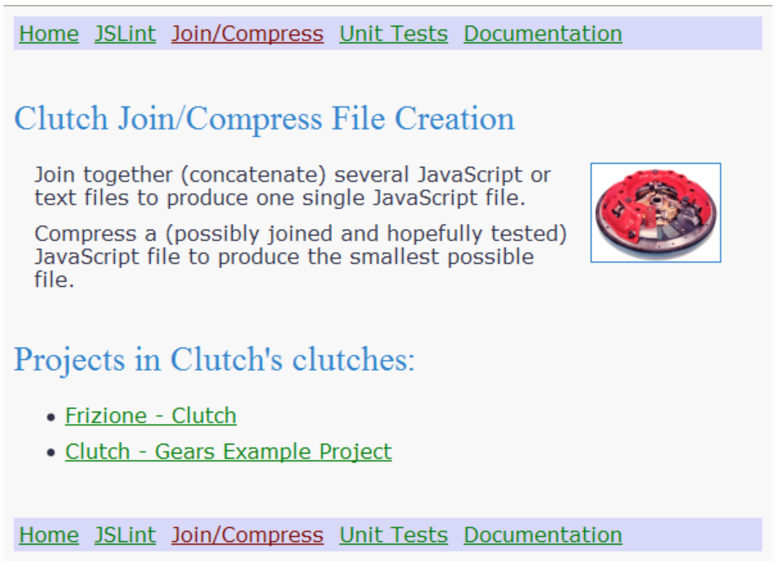
Indentation 4 Predefined (, separated)

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

Home JSLint Join/Compress Unit Tests Documentation

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

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

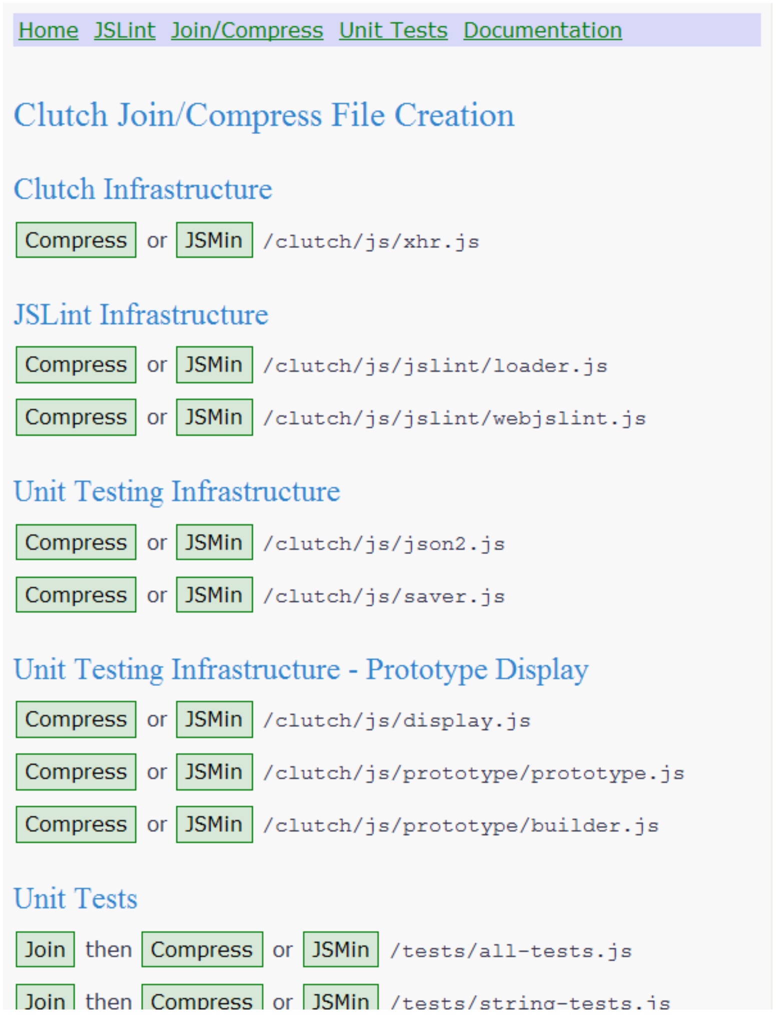
Compress a (possibly joined and hopefully tested) JavaScript file to produce the smallest possible file.

Projects in Clutch's clutches:

- Frizione - Clutch
- Clutch - Gears Example Project

Home JSLint Join/Compress Unit Tests Documentation

The main Join/Compress HTML page provides links to each project's Join/Compress page (if specified in the project's `clutch.json` file).


Home JSLint Join/Compress Unit Tests Documentation

Clutch Join/Compress File Creation

Clutch Infrastructure

Compress or JSMin /clutch/js/xhr.js

JSLint Infrastructure

Compress or JSMin /clutch/js/jslint/loader.js

Compress or JSMin /clutch/js/jslint/webjslint.js

Unit Testing Infrastructure

Compress or JSMin /clutch/js/json2.js

Compress or JSMin /clutch/js/saver.js

Unit Testing Infrastructure - Prototype Display

Compress or JSMin /clutch/js/display.js

Compress or JSMin /clutch/js/prototype/prototype.js

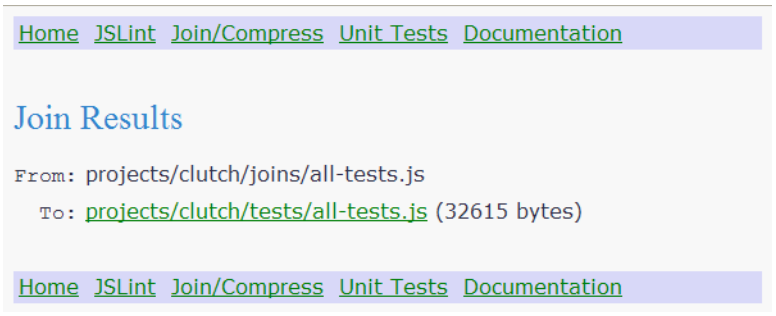
Compress or JSMin /clutch/js/prototype/builder.js

Unit Tests

Join then Compress or JSMin /tests/all-tests.js

Join then Compress or JSMin /tests/string-tests.js

Each individual project page contains links for the joining (concatenation) or compression of the principal JavaScript files that make up that project.

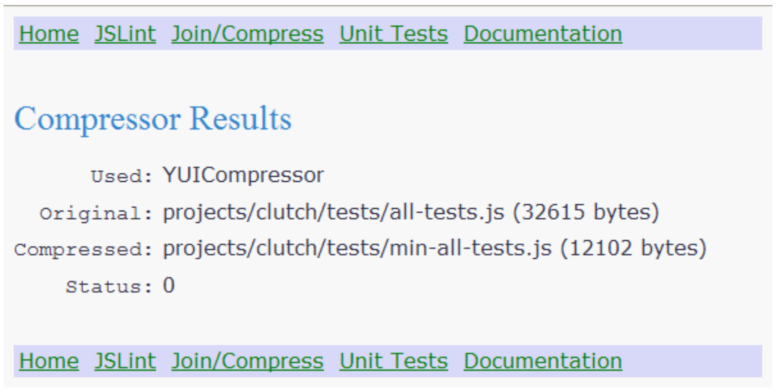

Home JSLint Join/Compress Unit Tests Documentation

Join Results

```
From: projects/clutch/joins/all-tests.js
  To: projects/clutch/tests/all-tests.js (32615 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.

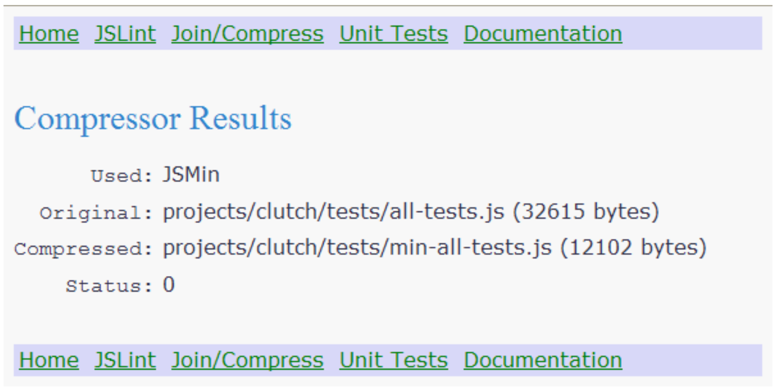

Home JSLint Join/Compress Unit Tests Documentation

Compressor Results

```
      Used: YUICompressor
  Original: projects/clutch/tests/all-tests.js (32615 bytes)
Compressed: projects/clutch/tests/min-all-tests.js (12102 bytes)
    Status: 0
```

Home JSLint Join/Compress Unit Tests Documentation

Each Compress command produces a results page indicating both the original JavaScript file byte size, and the compressed JavaScript file byte size.


Home JSLint Join/Compress Unit Tests Documentation

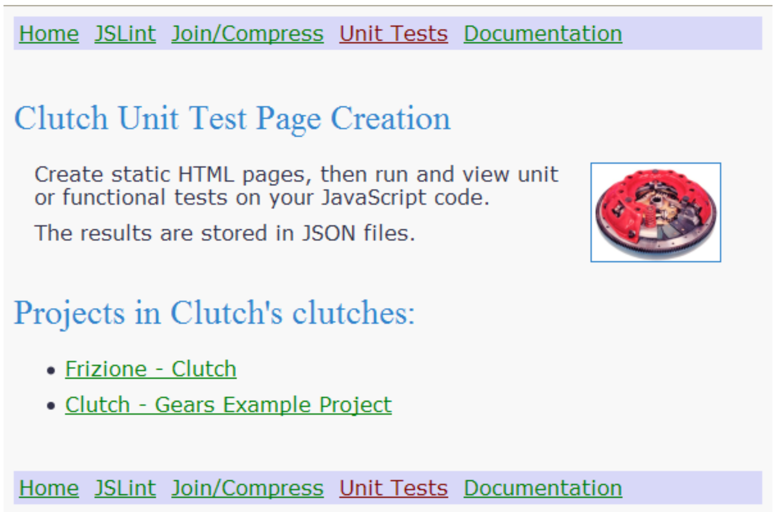
Compressor Results

```
      Used: JSMin
  Original: projects/clutch/tests/all-tests.js (32615 bytes)
Compressed: projects/clutch/tests/min-all-tests.js (12102 bytes)
    Status: 0
```

Home JSLint Join/Compress Unit Tests Documentation

Each JSMin command produces a results page indicating both the original JavaScript file byte size, and the compressed JavaScript file byte size.

The Unit Tests Page



Home JSLint Join/Compress Unit Tests Documentation

Clutch Unit Test Page Creation

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

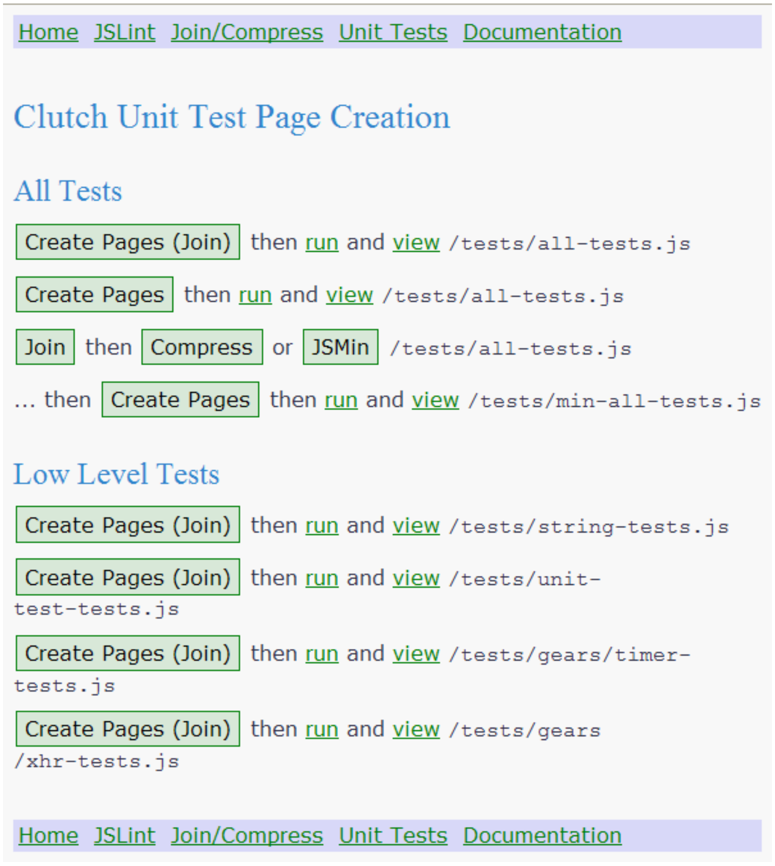
The results are stored in JSON files.

Projects in Clutch's clutches:

- Frizione - Clutch
- Clutch - Gears Example Project

Home JSLint Join/Compress Unit Tests Documentation

The main Unit Tests HTML page provides links to each project's Unit Test page (if specified in the project's `clutch.json` file).



Home JSLint Join/Compress Unit Tests Documentation

Clutch Unit Test Page Creation

All Tests

Create Pages (Join) then run and view /tests/all-tests.js

Create Pages then run and view /tests/all-tests.js

Join then Compress or JSMin /tests/all-tests.js

... then Create Pages then run and view /tests/min-all-tests.js

Low Level Tests

Create Pages (Join) then run and view /tests/string-tests.js

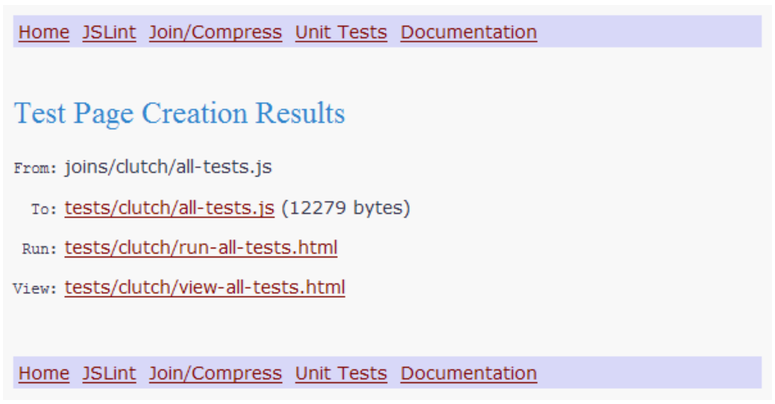
Create Pages (Join) then run and view /tests/unit-test-tests.js

Create Pages (Join) then run and view /tests/gears/timer-tests.js

Create Pages (Join) then run and view /tests/gears/xhr-tests.js

Home JSLint Join/Compress Unit Tests Documentation

Each individual project page contains links both for the creation of the static HTML test pages, and for running or viewing the results.


Home JSLint Join/Compress Unit Tests Documentation

Test Page Creation Results

From: joins/clutch/all-tests.js

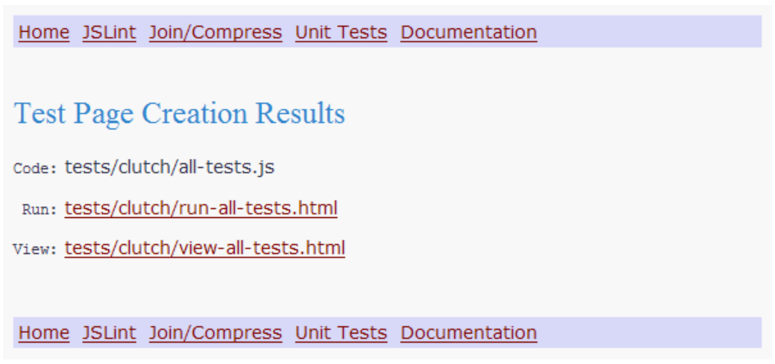
To: tests/clutch/all-tests.js (12279 bytes)

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

View: tests/clutch/view-all-tests.html

Home JSLint Join/Compress Unit Tests Documentation

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).


Home JSLint Join/Compress Unit Tests Documentation

Test Page Creation Results

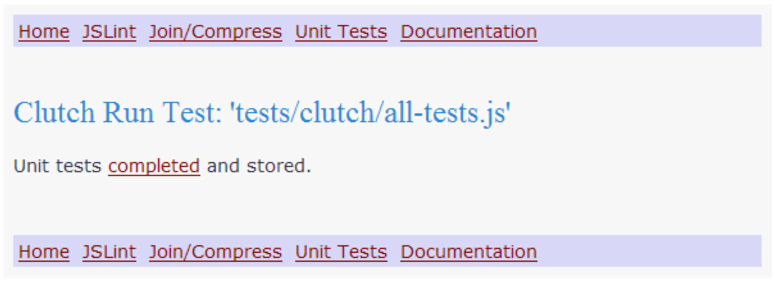
Code: tests/clutch/all-tests.js

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

View: tests/clutch/view-all-tests.html

Home JSLint Join/Compress Unit Tests Documentation

This example uses the JavaScript file 'as-is', again producing run tests and view results links.

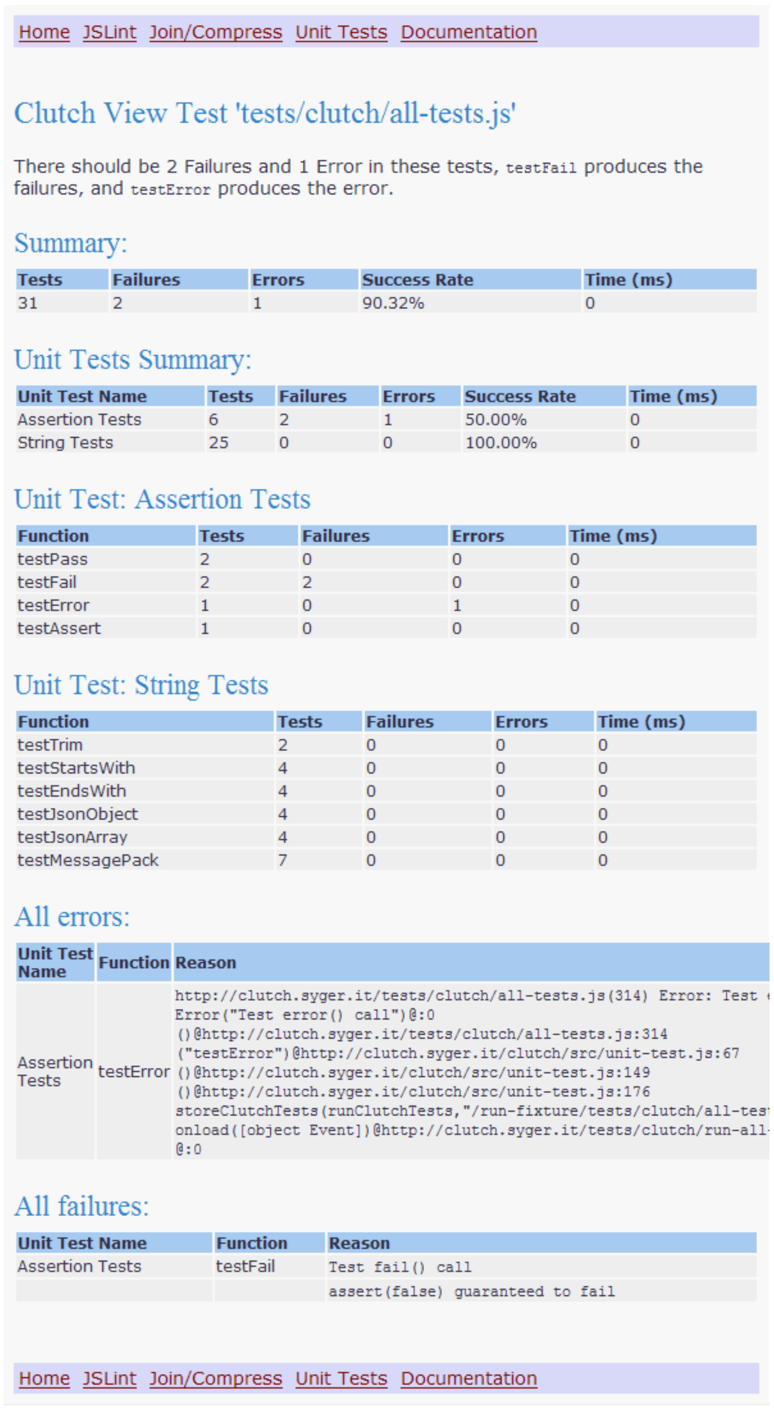

Home JSLint Join/Compress Unit Tests Documentation

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

Unit tests completed and stored.

Home JSLint Join/Compress Unit Tests Documentation

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.



Home JSLint Join/Compress Unit Tests Documentation

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`

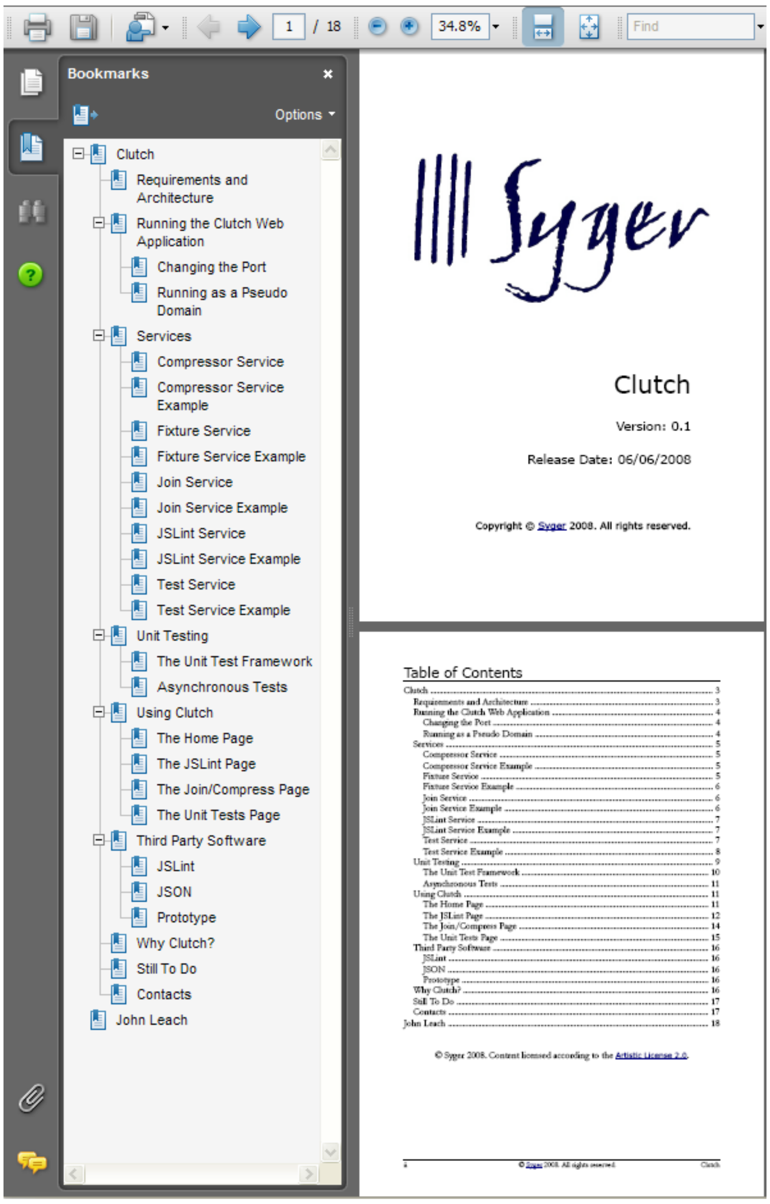
All failures:

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

Home JSLint Join/Compress Unit Tests Documentation

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

Documentation


1 / 18 34.8% Find

Bookmarks

Options

- Clutch
  - Requirements and Architecture
  - Running the Clutch Web Application
    - Changing the Port
    - Running as a Pseudo Domain
  - Services
    - Compressor Service
    - Compressor Service Example
    - Fixture Service
    - Fixture Service Example
    - Join Service
    - Join Service Example
    - JSLint Service
    - JSLint Service Example
    - Test Service
    - Test Service Example
  - Unit Testing
    - The Unit Test Framework
    - Asynchronous Tests
  - Using Clutch
    - The Home Page
    - The JSLint Page
    - The Join/Compress Page
    - The Unit Tests Page
  - Third Party Software
    - JSLint
    - JSON
    - Prototype
  - Why Clutch?
  - Still To Do
  - Contacts
- John Leach

Syger

Clutch

Version: 0.1

Release Date: 06/06/2008

Copyright © Syger 2008. All rights reserved.

Table of Contents

Clutch 3
Requirements and Architecture 3
Running the Clutch Web Application 4
Changing the Port 4
Running as a Pseudo Domain 4
Services 5
Compressor Service 5
Compressor Service Example 5
Fixture Service 5
Fixture Service Example 6
Join Service 6
Join Service Example 6
JSLint Service 7
JSLint Service Example 7
Test Service 7
Test Service Example 8
Unit Testing 9
The Unit Test Framework 10
Asynchronous Tests 11
Using Clutch 11
The Home Page 11
The JSLint Page 12
The Join/Compress Page 14
The Unit Tests Page 15
Third Party Software 16
JSLint 16
JSON 16
Prototype 16
Why Clutch? 16
Still To Do 17
Contacts 17
John Leach 18

© Syger 2008. Content licensed according to the Artistic License 2.0.

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**.

The Clutch Library

by John Leach

The Clutch library is a small set of JavaScript files which provide additional generic and Gears specific functionality. This document presents a brief overview of the library functions.

Browser

Source: `/Frizione/projects/clutch/src/browser.js`

Namespace: `clutch.browser`

Contains flags which test the presence of `IE`, `Opera`, `Webkit`, `Gecko`, `MobileSafari` and `Gears`.

Introspect

Source: `/Frizione/projects/clutch/src/introspect.js`

Namespace: `clutch`

`clutch.typeOf(obj) string`

Slightly more precise version of the `typeof` keyword.

- `object` - the object to check.

Returns the object type as a string.

`clutch.introspect(name, obj, indent, levels) string`

Creates a string representation of an object.

- `name` - the name of the object,
- `obj` - the object to introspect,
- `indent` - the optional start indentation, something like `"    "`, defaults to `""`,
- `levels` - optional, how many levels to drill down to, helps avoid recursion - default is `1`.

Returns a formatted string representation of the object.

String

Source: `/Frizione/projects/clutch/src/string.js`

Namespace: `clutch.date` and `clutch.string`

`clutch.date.toStandardJSON()`

Uses 'standard' formatting for `Date` objects when converting to JSON.

`clutch.date.toMicrosoftJSON()`

Uses 'Microsoft' formatting for `Date` objects when converting to JSON.

`clutch.date.toClutchJSON()`

Uses 'Microsoft' formatting with milliseconds for `Date` objects when converting to JSON.

`clutch.string.trim(string) string`

Removes whitespace at the beginning and end of a string.

- `string` - the string to trim.

Returns the modified string.

`clutch.string.startsWith(string, match) boolean`

Checks if a string starts with a specified substring.

- `string` - the string to check,
- `match` – the substring to match.

Returns `true` if the string starts with the substring, otherwise `false`.

`clutch.string.endsWith(string, match) boolean`

Checks if a string ends with a specified substring.

- `string` - the string to check,
- `match` – the substring to match.

Returns `true` if the string ends with the substring, otherwise `false`.

`clutch.string.toJSON(object) string`

Converts an object to a JSON formatted string.

- `object` - the object to convert.

Returns the converted string. Requires either the Prototype or JSON libraries.

`clutch.string.fromJSON(string) object`

Converts a JSON formatted string to an object.

- `string` - the string to convert.

Returns the converted object. Requires either the Prototype or JSON libraries.

Gears

Source: `/Frizione/projects/clutch/src/gears/gears.js`

Namespace: `clutch`

`clutch.isGearsInstalled() boolean`

Returns `true` if Gears is installed, otherwise `false`.

`clutch.gearsFactory() Factory`

Gets the Gears `Factory` object.

`clutch.createGearsDatabase() Database`

Creates a new Gears `Database` object.

`clutch.createGearsDesktop() Desktop`

Creates a new Gears `Desktop` object.

`clutch.createGearsHttpRequest() HttpRequest`

Creates a new Gears `HttpRequest` object.

`clutch.createGearsLocalServer() LocalServer`

Creates a new Gears `LocalServer` object.

`clutch.createGearsTimer() Timer`

Creates a new Gears `Timer` object.

`clutch.createGearsWorkerPool() WorkerPool`

Creates a new Gears `WorkerPool` object.

Timer

Source: `/Frizione/projects/clutch/src/gears/timer.js`

Namespace: `clutch.timer`

`clutch.timer.setTimeout(code, millis) number`

Creates a timeout that will call a code fragment or function after a specific period of time has elapsed.

- `code` - the code fragment or function to call,
- `millis` – the number of milliseconds to wait.

Returns the identifier of the timeout.

`clutch.timer.setInterval(code, millis) number`

Creates an interval that will call a code fragment or function repeatedly after a specific period of time has elapsed.

- `code` - the code fragment or function to call,
- `millis` – the number of milliseconds to wait.

Returns the identifier of the interval.

`clutch.timer.clearTimeout(identifier)`

Cancels a timeout before it is executed.

- `identifier` – the identifier of the timeout to cancel.

`clutch.timer.clearInterval(identifier)`

Cancels an interval.

- `identifier` – the identifier of the interval to cancel.

XHR

Source: `/Frizione/projects/clutch/src/gears/xhr.js`

Namespace: `clutch.xhr`

`clutch.xhr.executeRequest(method, url, params, body, timeout, handler)`

Executes an HttpRequest.

- `method` - either `"GET"` or `"POST"`,
- `url` - the absolute URL to get or post to,
- `params` - optional parameters, do your own value encoding though,
- `body` - optional body for posts,
- `timeout` - the optional maximum amount of time to wait for a reply,
- `handler` – the function which handles the response.

The `handler` function receives three arguments:

- `status` – the response status code,

- `statusText` – the response status text,
- `responseText` – the response text.

WorkerPool Messages

Source: `/Frizione/projects/clutch/src/gears/wp-messages.js`

Namespace: `clutch.wp`

clutch.wp.handlers object

The `clutch.wp.handlers` object can contain a series of properties which represent a command name, and a function which handles the named command. Each function receives the `message` object as its only argument.

The `message` object `body` property must contain a `command` property with the command value as a string.

See `/Frizione/projects/clutch/src/gears/wp-unit-test.js` for an example.

clutch.wp.onMessage(depr1, depr2, message)

The `WorkerPool onmessage` handler. Passes the message to a handler function based on the command value.

- `depr1` – the deprecated message contents,
- `depr2` – the deprecated ID of the source worker,
- `message` – the object containing all information about the message.

See `/Frizione/projects/clutch/src/gears/wp-unit-test.js` for an example.

Database Utilities

Source: `/Frizione/projects/clutch/src/gears/db-utils.js`

Namespace: `clutch.db`

clutch.db.fromRow(result, columns) object

Takes a ResultSet, which is expected to contain zero or one results, and converts it to an object.

- `result` – the ResultSet,
- `columns` – an array of column names.

Returns the object, with properties equal to the column names, and values from the ResultSet, or `null` if there where no results.

`clutch.db.fromRows(result, columns) array`

Takes a ResultSet, which is expected to contain zero or more results, and converts it to an array of objects.

- `result` – the ResultSet,
- `columns` – an array of column names.

Returns the array of objects, each with properties equal to the column names, and values from the ResultSet, or `null` if there were no results.

`clutch.db.optionalQuery(params)`

Constructs the query constraints using an object with optional properties.

- `params` – the parameter object, accepted properties are:
  - `where` – the `WHERE` clause,
  - `groupBy` – the `GROUP BY` clause,
  - `having` – the `HAVING` clause,
  - `orderBy` – the `ORDER BY` clause,
  - `limit` – the `LIMIT` clause,
  - `offset` – the `OFFSET` clause.

Returns the query constraint as a string.

Database Logger

Source: `/Frizione/projects/clutch/src/gears/db-logger.js`

Namespace: `clutch.db`

`clutch.db.logger(name) DatabaseLogger`

Opens a Database object with the specified name, and creates a `clutch_logger` table, if one does not already exist.

- `name` – the database name.

`DatabaseLogger.log(name, value) number`

Adds a log record with the given name and value.

- `name` – the name (maximum 256 characters),
- `value` – the value (maximum 4096 characters).

Returns the number of rows affected.

`DatabaseLogger.get(id) object`

Gets the object with the specified identifier.

- `id` – the record identifier.

Returns the object, or `null` if not found.

`DatabaseLogger.list(params) array`

Returns an array of objects, using optional query constraints.

- `params` – the optional query constraints.

Returns the array, or `null` if none found.

`DatabaseLogger.remove(id) number`

Removes (deletes) the record with the specified identifier.

- `id` – the record identifier.

Returns the number of rows affected.

`DatabaseLogger.removeAll() number`

Removes (deletes) all records from the table.

Returns the number of rows affected.

Clutch RoadMap

by John Leach

Now that the basic infrastructure for Clutch is complete (the primary target was unit testing in a Gears `WorkerPool`), the time has come to take a critical look at what has been created, take note of the good parts, and rework the bad parts.

These are currently just musings, based on my own experience of using Clutch, and there is no guarantee that anything written here will see the light of day as working code. One important point though, I'm still prepared to make breaking changes 'for the common good'. They won't be gratuitous, I'm not that masochistic.

I'm currently treating this document as a reminder to myself that things can be improved. The writing style will probably reflect that statement.

The Good Parts

I'm happy with the unit testing code. Although there are already many JavaScript unit testing frameworks on the market, and this is yet another, I still think it has a few good points:

- It actually works,
- It is a small(ish) amount of code – about 18KB,
- It stores the results as an external JSON file,
- Er, that's it.

I'd like to improve (reduce the amount of) the code that's already there, but it's not very high on my list of priorities.

The JSLint service is a great help. I always check that my code is JSLint clean before running any tests. It's not foolproof, but it certainly acts like a 'missing' compiler. The feeling is definitely warm and fuzzy.

The Join service is just great for building up specialised libraries from small individual modules. I can't cope with 50-100KB JavaScript source files, but I don't want a dozen script tags in my HTML pages either. Plus `WorkerPools` accept only one URL.

The projects directory works nicely, using a simple JSON configuration file. I'm happy with that idea, and I think it can be used elsewhere too.

The So-So Parts

After a few days of use, I'm finding that building those little form tags for each damn operation is just too manual. On the plus side, it does actually work.

I keep forgetting to click on the right Join button after making some small modification to one of the source files. This will only get worse as the number of files grows.

I'd like to see a JSON output for the JSLint reports too. I'm thinking ahead here, towards something with a CruiseControl style to it. I really want a 'check everything – twice' system, especially before a source control commit.

The Bad Parts

The whole build process is just too manual. I'm getting fed up with my 'click on a button' solution in half a dozen web pages. It's just not working. I forget to do things. Nothing is telling me what I forgot to do.

So I'm going to have to automate a lot of this functionality. I have a few ideas.

The Missing Parts

Well, JavaScript is not such a bad language after all, and Gears is becoming a useful platform.

I think the next software development cycle is going to have to tackle the object relational management problem. Right now I'm using a ODBC/JDBC type model – the DatabaseLogger is an example. I'll get very tired of that after a dozen tables or so.

Frizione – Clutch Version History

by John Leach

This section notes changes made in the various version releases, in reverse chronological order.

Version 0.3 – 26/06/2008

This is an important milestone as Clutch can now perform unit testing within a `WorkerPool`. The unit tests have been extended to include Gears WorkerPool, Gears Database, Gears Timer, and Gears XHR testing. The documentation has been updated to reflect most changes.

Minor bug fixes and modifications were made to the unit testing framework, including a message based protocol for remote WorkerPool testing and reporting.

Version 0.2 - 18/06/2008

Added JSMin (the Ruby version), to provide some compression functionality to those who don't want to install Java on their computer.

Added `/projects` directory for user projects which won't be disturbed when updating the Clutch Subversion repository. Migrated Clutch JavaScript code to `/projects/clutch` (Clutch – the library, is the first project for Clutch – the framework, dog food, etc).

Completed the first (working) model for asynchronous unit testing.

Version 0.1.1 – 10/06/2008

Never make changes at the last minute without running all your unit tests. Especially if it is 1 am in the morning. Such is life, and breaking this golden rule invokes numerous laws of Murphy, one of which was that the server stopped working. Fixed within half an hour, fortunately.

Version 0.1 - 10/06/2008

The first public release. Unfortunately, I had to change the project name at the last moment, since `clutch` was already being used on the Google Code web site – hence the new name `frizione` (which is Clutch in Italian).

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.