Frizione - Clutch

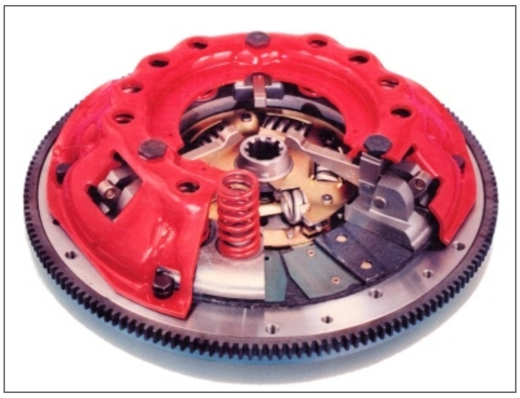
Version: 0.4

Release Date: 16/07/2008

Table of Contents

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, Installation 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 now written in JavaScript, using the Helma JavaScript Web Server, and Java. To use Clutch you will need to install Helma (I'm using version 1.6.2) and install Java (I'm using version 1.6.0).

First, install Helma and Java if not already present on your computer. The Helma download page provides archived files for Linux, Mac OS X, and Windows, and installation instructions.

Once you have correctly installed Helma, create a directory `/Frizione-0.4` with a sub directory `/Frizione`, and copy the contents of the Helma directory into `/Frizione-0.4/Frizione`. Then extract the archived file (or SVN repository) into `/Frizione-0.4`. That may seem a little complex, but Frizione makes a few subtle changes to the standard Helma distribution, so this process effectively adds the Frizione web application, but also overwrites a few Helma files too.

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

Service URL	Description
`/jslint`	Runs JSLint on a specific JavaScript file.
`/jsonview`	Displays a JSON file contents.
`/htmlview`	Displays (redirects to) a static HTML file.
`/cssjoin`	Joins or concatenates a series of CSS files into a single file.
`/jsjoin`	Joins (concatenates) a series of JavaScript files into a single file.
`/cssminify`	Minifies (compresses) a CSS file.
`/jsminify`	Minifies (compresses) a JavaScript file
`/jstest`	Runs a unit test JavaScript file.
`/jsontest`	Displays a unit test JSON formatted results file.
`/readfixture`	Provides GET (read) operations for fixture (unit testing data) files.
`/writefixture`	Provides POST (write) operations for fixture (unit testing data) files.

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 modified Helma web server running. To start the web server, open a command prompt in the `/Frizione-0.4/Frizione` directory, then type:

```
start
```

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

```
Starting HTTP server on port 80
Serving applications from C:\Frizione-0.4\Frizione\
Starting Helma 1.6.2 (April 7 2008) on Java 1.6.0_03
```

That's it, Clutch is up and running. You can stop the web server 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 `start.bat` or `start.sh` script. Open the file in your favourite text editor, where you should see (`start.bat`):

```
...
:: Set TCP ports for Helma servers
:: (comment/uncomment to de/activate)
set HTTP_PORT=80
rem set XMLRPC_PORT=8081
rem set AJP13_PORT=8009
rem set RMI_PORT=5050
```

or (`start.sh`)

```
...
# Set TCP ports for Helma servers
# (comment/uncomment to de/activate)
HTTP_PORT=80
# XMLRPC_PORT=8081
# AJP13_PORT=8009
# RMI_PORT=5050
```

Change the `HTTP_PORT` setting to the port value that you want, then restart the web 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 web server running, open your browser and type the URL `http://clutch.syger.it/` which should now present you with the Helma home page. Typing the URL `http://clutch.syger.it/frizione/` should present you with the Frizione home page.

Services

The Frizione web application provides a suite of services which aid in the development, testing, and deployment of JavaScript software. These services are made automatically available, based on a small, simple set of file naming conventions.

Frizione searches the `apps`, `frizione-projects` and `modules` sub directories of the web server, looking for a `frizione.json` file which (currently) contains a single name element:

```
{
    "name":   "Clutch Library"
}
```

Only those sub directories containing a valid `frizione.json` file will be included for special treatment by Clutch.

The `apps` and `modules` sub directories are specific to Helma web application development, and currently an experimental reduced set of Clutch services are available. The `frizione-projects` sub directory, on the other hand, is designed specifically for client side JavaScript development, and is explained in detail in the following sections.

JSLint Service (/jslint)

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 running JSLint on any JavaScript file ending in `.js`, except those ending in `.join.js`, `.min.js`, or `.test.js`., which are treated differently. Clutch also creates a `jslint.options.json` file which provides project wide JSLint options settings. This file can be modified using your favourite text editor.

To invoke the service, select the "`JSLint files`" link under "`JavaScript files`" in the desired project page (see `http://clutch.syger.it/frizione/projects/clutch/` for an example), then select a file to check. Clutch will then run JSLint on the JavaScript file, using the `jslint.options.json` values.

You can also make file specific settings by creating special comments inside the JavaScript file itself. Use `/*jslint ... */` to make specific options settings, and `/*global ... */` to define specific global variables. See `http://clutch.syger.it/frizione/projects/clutch/jslint/js/dev/browser.js` for an example.

JSON file view (/jsonview)

Clutch will display the contents of a JSON file ending in `.json`, except those ending in `.test.json`, which are treated with greater respect.

To invoke the service, select the "`Viewable files`" link under "`JSON files`" in the desired project page (see `http://clutch.syger.it/frizione/projects/clutch/` for an example), then select a file to view. Clutch will then display the contents of the JSON file.

HTML file view (/htmlview)

Clutch will display the contents of a HTML file ending in `.html`.

To invoke the service, select the "`Viewable files`" link under "`Static files`" in the desired project page (see `http://clutch.syger.it/frizione/projects/clutch-gears/` for an example), then select a file to display. Clutch will then display (actually redirect to) the

HTML file.

CSS Join Service (/cssjoin)

The join (or concatenate) service can be run on any CSS file that ends with `.join.css`. The actual join parameters are specified inside the CSS file in a special `/*join ... */` comment. Clutch checks that you supply an output URL, and that it is not identical to the input URL. Clutch then joins together a list of CSS 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, select the "`Join files`" link under "`CSS files`" in the desired project page (see http://clutch.syger.it/frizione/projects/clutch/ for an example), then select a file to join. Clutch will then execute and display the results of the join operation.

CSS Join Service Example

This example is taken from `/Frizione-0.4/Frizione/frizione-projects/clutch/css/dev-browser/clutch.join.css`.

```
/*join to: /css/clutch-all.css
     gzip: /css/clutch-all.css.gz */
<%= include('./clutch.css', './jslint.css') %>
```

Given the following directory layout:

```
/Frizione-0.4
  /Frizione
    /frizione-projects
      /clutch
        /css
          /dev-browser
            clutch.join.css
            clutch.css
            jslint.css
          clutch-all.css
          clutch-all.css.gz
        ...
        frizione.json
```

The relative URL to the two included files is `./`. Note also that the `/*join ... */` command comment requires each parameter on a separate line.

CSS join parameters:

Name	Value
`to`	The destination file URL within the project directory.
`gzip`	The optional destination gzip file URL within the project directory.

See also `/Frizione-0.4/Frizione/frizione-projects/clutch/css/dev-browser/clutch.join.css`.

JavaScript Join Service (/jsjoin)

The join (or concatenate) service can be run on any JavaScript file that ends with `.join.js`. The actual join parameters are specified inside the JavaScript file in a special `/*join ... */` comment. Clutch then joins together a list of JavaScript 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, select the "`Join files`" link under "`JavaScript files`" in the desired project page (see http://clutch.syger.it/frizione/projects/clutch/ for an example), then select a file to join. Clutch will then execute and display the results of the join operation.

JavaScript Join Service Example

This example is taken from `/Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-browser/browser-unittester.join.js`.

```
/*join to: /js/browser-unittester.cjs
     gzip: /js/browser-unittester.cjs.gz */
<%= include('./json2.js', './xhr.js', '../dev/unit-test.js', './saver.js') %>
```

Given the following directory layout:

```
/Frizione-0.4
  /Frizione
    /frizione-projects
      /clutch
        /js
          /dev
            unit-test.js
          /dev-browser
            browser-unittester.join.js
            json2.js
            xhr.js
            saver.js
          browser-unittester.cjs
          browser-unittester.cjs.gz
        ...
        frizione.json
```

The relative URL to the four included files is `./` and `../dev` respectively. Note also that the `/*join ... */` command comment requires each parameter on a separate line.

JavaScript join parameters:

Name	Value
`to`	The destination file URL within the project directory.
`gzip`	The optional destination gzip file URL within the project directory.

See also `/Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-browser/browser-unittester.join.js`.

CSS Minify Service (/cssminify)

The minify or compressor service can be run on any CSS file that ends with `.join.css`. The actual minify parameters are specified inside the CSS file in a special `/*minify ... */` comment. Clutch checks that you supply an output URL, and that it is not identical to the input URL. Clutch then optionally joins together a list of CSS files, producing a single concatenated file, which is then minified using YUICompressor.

To invoke the service, select the "`Minify files`" link under "`CSS files`" in the desired project page (see http://clutch.syger.it/frizione/projects/clutch/ for an example), then select a file to minify. Clutch will then execute and display the results of the minify operation.

CSS Minify Service Example

This example is taken from `/Frizione-0.4/Frizione/frizione-projects/clutch/css/dev-browser/clutch.min.css`.

```
/*minify to: /css/clutch-all.css
       gzip: /css/clutch-all.css.gz */
<%= include('./clutch.css', './jslint.css') %>
```

Given the following directory layout:

```
/Frizione-0.4
  /Frizione
    /frizione-projects
      /clutch
        /css
          /dev-browser
            clutch.join.css
            clutch.css
            jslint.css
          clutch-all.css
          clutch-all.css.gz
        ...
        frizione.json
```

The relative URL to the two included files is `./`. Note also that the `/*minify ... */` command comment requires each parameter on a separate line.

CSS minify parameters:

Name	Value
`to`	The destination file URL within the project directory.
`gzip`	The optional destination gzip file URL within the project directory.
`mc`	The optional maximum number of characters before a forced line break (default is 0).

See also `/Frizione-0.4/Frizione/frizione-projects/clutch/css/dev-browser/clutch.min.css`.

JavaScript Minify Service (/jsminify)

The minify or compressor service can be run on any JavaScript file that ends with `.min.js`. The actual minify parameters are specified inside the JavaScript file in a special `/*minify ... */` comment. Clutch checks that you supply an output URL, and that it is not identical to the input URL. Clutch then optionally joins together a list of JavaScript files, producing a single concatenated file, which is then minified using YUICompressor.

To invoke the service, select the "`Minify files`" link under "`JavaScript files`" in the desired project page (see `http://clutch.syger.it/frizione/projects/clutch/` for an example), then select a file to minify. Clutch will then execute and display the results of the minify operation.

JavaScript Minify Service Example

This example is taken from `/Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-browser/browser-unittester.min.js`.

```
/*minify to: /js/browser-unittester.cjs
       gzip: /js/browser-unittester.cjs.gz */
<%= include('./json2.js', './xhr.js', '../dev/unit-test.js', './saver.js') %>
```

Given the following directory layout:

```
/Frizione-0.4
  /Frizione
    /frizione-projects
      /clutch
        /js
          /dev
            unit-test.js
          /dev-browser
            browser-unittester.join.js
            json2.js
            xhr.js
            saver.js
          browser-unittester.cjs
          browser-unittester.cjs.gz
        ...
        frizione.json
```

The relative URL to the four included files is `./` and `../dev` respectively. Note also that the `/*minify ... */` command comment requires each parameter on a separate line.

JavaScript minify parameters:

Name	Value
`to`	The destination file URL within the project directory.
`gzip`	The optional destination gzip file URL within the project directory.
`mc`	The optional maximum number of characters before a forced line break (default is 0).
`munge`	Optionally reduce function and variable names (default is `false`).

ps	Optional, preserve semicolons (default is true).
opt	Optional, use micro optimisations (default is true).

See also /Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-browser/browser-unittester.min.js.

JavaScript Test Service (/jstest)

The test service can be run on any JavaScript file that ends with .test.js. The actual test parameters are specified inside the JavaScript file in a special /*test ... */ comment. Clutch checks that you supply an output URL, and that it is not identical to the input URL. Clutch then optionally joins together a list of JavaScript files, producing a single concatenated file, which is then optionally minified using YUICompressor, and the test is run.

To invoke the service, select the "Test files" link under "JavaScript files" in the desired project page (see http://clutch.syger.it/frizione/projects/clutch/ for an example), then select a file to test. Clutch will then execute and display the results of the test.

JavaScript Test Service Example

This example is taken from /Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-test/string.test.js.

```
/*test to: /js/test/string.js
     json: /js/test/string.test.json */
<%= this.include('../dev/string.js') %>
<%= this.include('./string.js') %>
```

Given the following directory layout:

```
/Frizione-0.4
  /Frizione
    /frizione-projects
      /clutch
        /js
          /dev
            string.js
          /dev-test
            string.test.js
            string.js
          /test
            string.js
            string.test.json
        ...
        frizione.json
```

The relative URL to the two included files is ./ and ../dev respectively. Note also that the /*test ... */ command comment requires each parameter on a separate line.

JavaScript test parameters:

Name	Value
to	The destination file URL within the project directory.

`gzip`	The optional destination gzip file URL within the project directory.
`json`	The destination file URL within the project directory of the unit tests. Make sure that the URL ends with `.test.json` so that Clutch will recognise the file as a unit test results file.
`type`	The optional testing environment type, one of `browser`, `gears`, or `workerpool` (default is `browser`).

See also `/Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-test/string.test.js`.

JSON Test Result View Service (/jsontest)

The test result view service can be run on any JSON file that ends with `.test.json`.

To invoke the service, select the "`Test results files`" link under "`JSON files`" in the desired project page (see `http://clutch.syger.it/frizione/projects/clutch/` for an example), then select a file to view. Clutch will then display the results of the test.

Read Fixture Service (/readfixture)

This service allows you to read data from hard disk. When sent a `GET`, the service reads data from the file specified in the URL, optionally modifying the contents with parameter values specified in the `GET` request.

To invoke the service, send a `GET` request to `/frizione/projects/`*project-name*`/readfixture/`, appending the absolute file path, with respect to the project root directory, as part of the URL. Clutch will read that file, accepting and executing `include` commands (see the Join Service, above), as well as injecting parameters into the constructed text file.

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

- any parameter names starting with `frizione`, are reserved by Clutch,
- within the text file each parameter value can be referenced by name,
- more complicated expressions can be achieved using JavaScript code snippets.

Read Fixture Service Example

The following example is taken from `/Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-test/gears/xhr.js`:

```
var abort = clutch.xhr.executeRequest("GET",
    '/frizione/projects/clutch/readfixture/js/dev-test/gears/xhr-test-data.json',
    null, null, 2000, this.validUrlHandler);
```

Write Fixture Service (/writefixture)

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.

To invoke the service, send a `POST` request to `/frizione/projects/`*project-name*`/writefixture/`, appending the absolute output file path, with respect to the project root directory, as part of the URL.

Write Fixture Service Example

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

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

  jsonUrl = '/frizione/projects/clutch/writefixture' + 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-0.4/Frizione/frizione-projects/clutch/js/dev-browser/saver.js` for the complete code example.

TODO: show true fixtures examples...

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 unobtrusive 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 the unit test framework to your unit test code, 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-0.4/Frizione/frizione-projects/clutch/js/dev-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, again 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-0.4/Frizione/frizione-projects/clutch/js/dev-tests/` for example unit test code. Additionally, `/Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-`

`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-0.4/Frizione/frizione-projects/clutch/js/dev-tests/unit-test.js` for an example of synchronous unit test code, and `/Frizione-0.4/Frizione/frizione-projects/clutch/js/dev-tests/gears/xhr.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 `/frizione-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 `/frizione-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 `/frizione-projects`,
- create a `frizione.json` file in that folder,
- click on "`Refresh project list`" in the Clutch home page.

The `frizione.json` file contains the project name. The following is the clutch project `frizione.json` file:

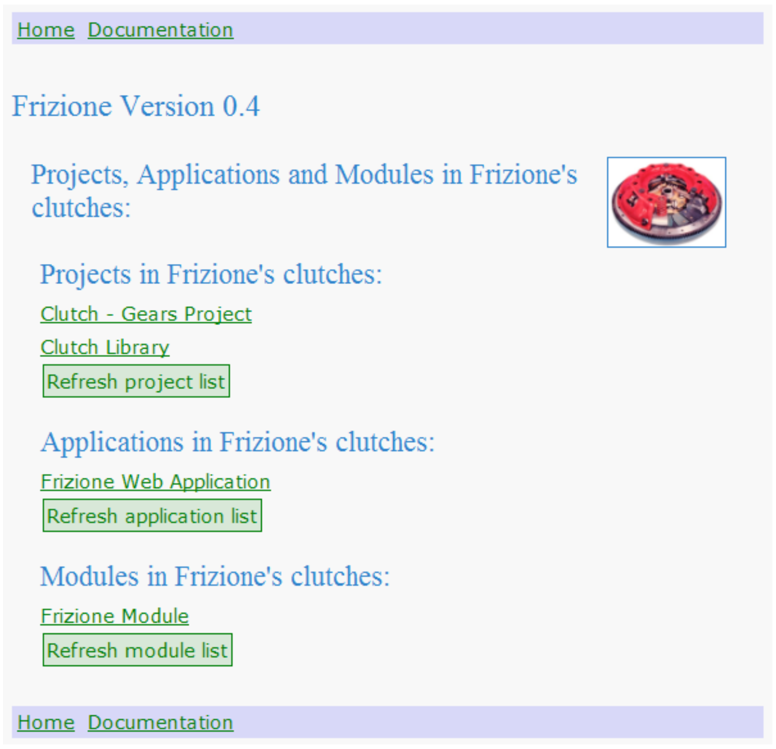
```
  {
    "name":   "Clutch Library"
  }
```

See `/Frizione-0.4/Frizione/frizione-projects/clutch` and `/Frizione-0.4/Frizione/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/frizione/`, you should then see the Clutch home page.

The Home Page


Home Documentation

Frizione Version 0.4

Projects, Applications and Modules in Frizione's clutches:

Projects in Frizione's clutches:

Clutch - Gears Project

Clutch Library

Refresh project list

Applications in Frizione's clutches:

Frizione Web Application

Refresh application list

Modules in Frizione's clutches:

Frizione Module

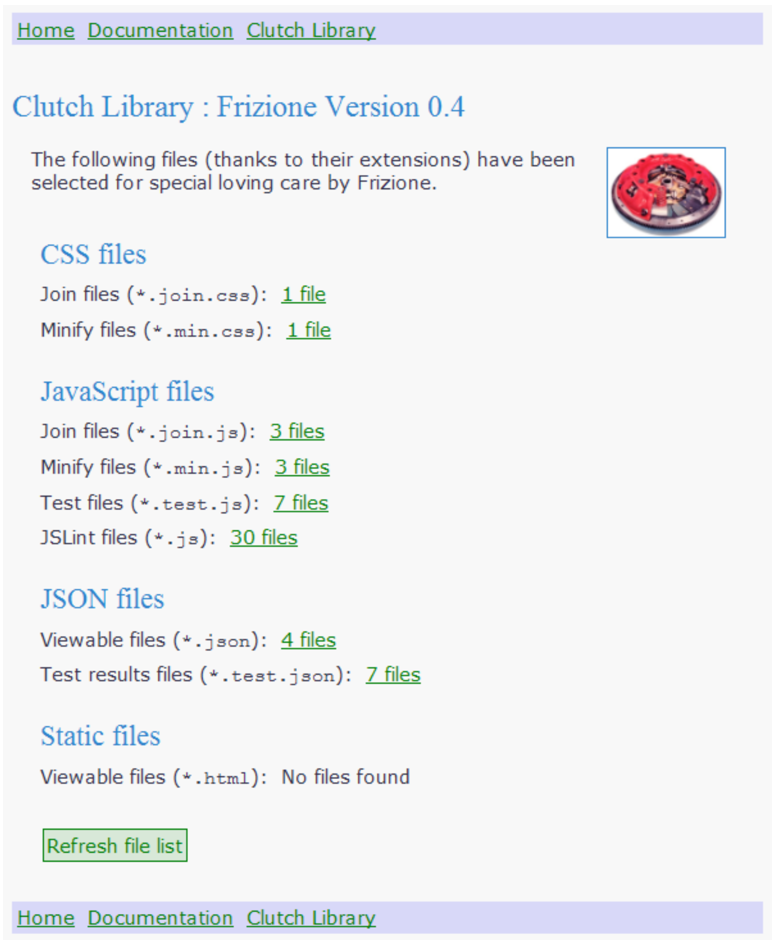
Refresh module list

Home Documentation

The home page provides access to the project pages. Clutch caches project information, so if you make changes, don't forget to click on one of the refresh buttons.

The Project Page

Once you have chosen a particular project, the main project page is displayed:


Home Documentation Clutch Library

Clutch Library : Frizione Version 0.4

The following files (thanks to their extensions) have been selected for special loving care by Frizione.

CSS files

Join files (`*.join.css`): 1 file

Minify files (`*.min.css`): 1 file

JavaScript files

Join files (`*.join.js`): 3 files

Minify files (`*.min.js`): 3 files

Test files (`*.test.js`): 7 files

JSLint files (`*.js`): 30 files

JSON files

Viewable files (`*.json`): 4 files

Test results files (`*.test.json`): 7 files

Static files

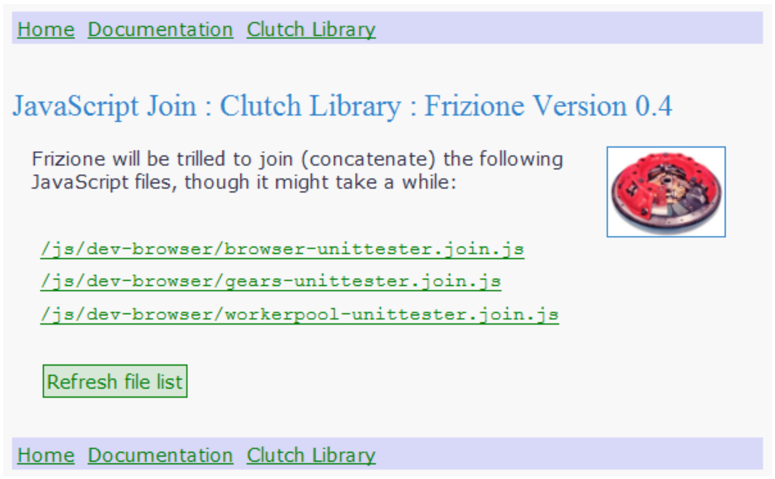
Viewable files (`*.html`): No files found

Refresh file list

Home Documentation Clutch Library

Again, Clutch caches project file information, so if you make changes, don't forget to click on the refresh button. Using the file extension naming convention, from this page you can access the Join, Minify, JSLint, Test and View services.

The Join Page


Home Documentation Clutch Library

JavaScript Join : Clutch Library : Frizione Version 0.4

Frizione will be trilled to join (concatenate) the following JavaScript files, though it might take a while:

/js/dev-browser/browser-unittester.join.js

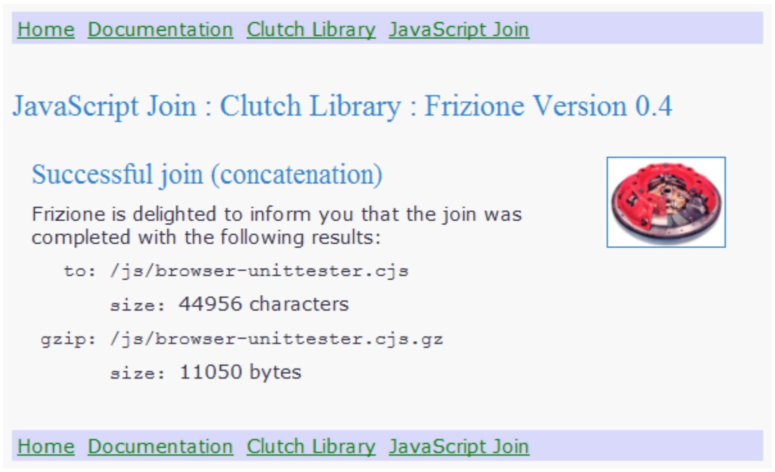
/js/dev-browser/gears-unittester.join.js

/js/dev-browser/workerpool-unittester.join.js

Refresh file list

Home Documentation Clutch Library

The main Join page provides links to each join file. Not to be repetitive, but Clutch caches project file information, so if you make changes, don't forget to click on the refresh button. Clicking on any of the displayed links executes the Join, producing a Join results page:



Home Documentation Clutch Library JavaScript Join

JavaScript Join : Clutch Library : Frizione Version 0.4

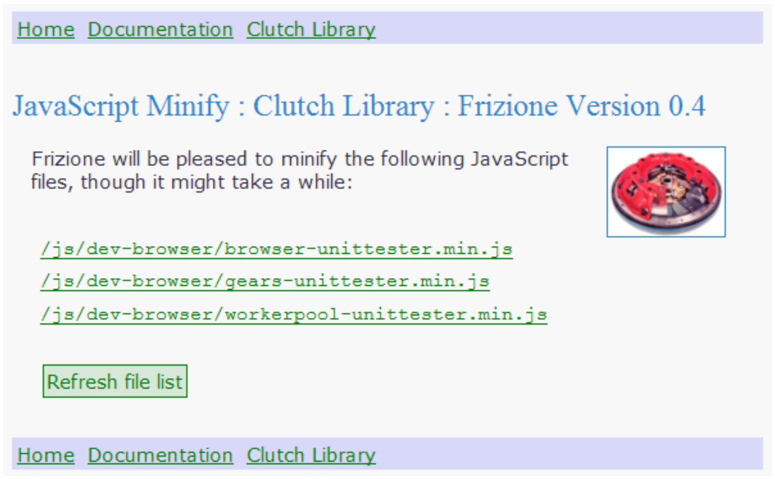
Successful join (concatenation)

Frizione is delighted to inform you that the join was completed with the following results:

```
  to: /js/browser-unittester.cjs
        size: 44956 characters
gzip: /js/browser-unittester.cjs.gz
        size: 11050 bytes
```

Home Documentation Clutch Library JavaScript Join

The Minify Page



Home Documentation Clutch Library

JavaScript Minify : Clutch Library : Frizione Version 0.4

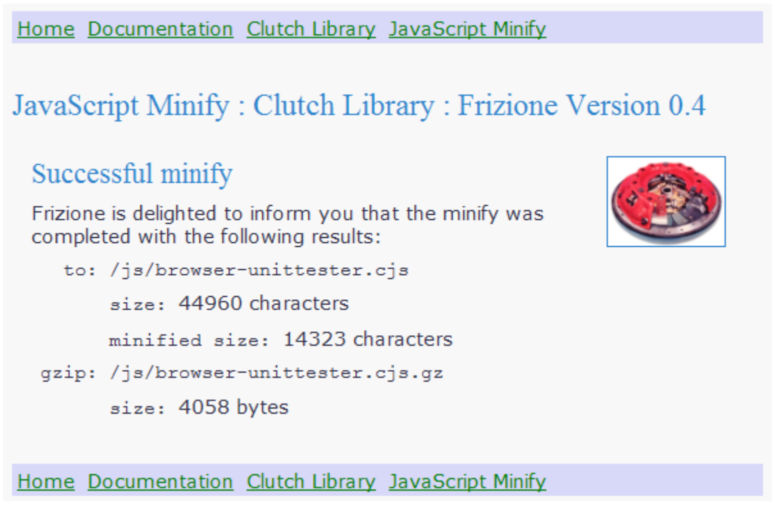
Frizione will be pleased to minify the following JavaScript files, though it might take a while:

/js/dev-browser/browser-unittester.min.js
/js/dev-browser/gears-unittester.min.js
/js/dev-browser/workerpool-unittester.min.js

Refresh file list

Home Documentation Clutch Library

The main Minify page provides links to each minify file. This might have been explained before, but Clutch caches project file information, so if you make changes, don't forget to click on the refresh button. Clicking on any of the displayed links executes the Minify, producing a Minify results page:



Home Documentation Clutch Library JavaScript Minify

JavaScript Minify : Clutch Library : Frizione Version 0.4

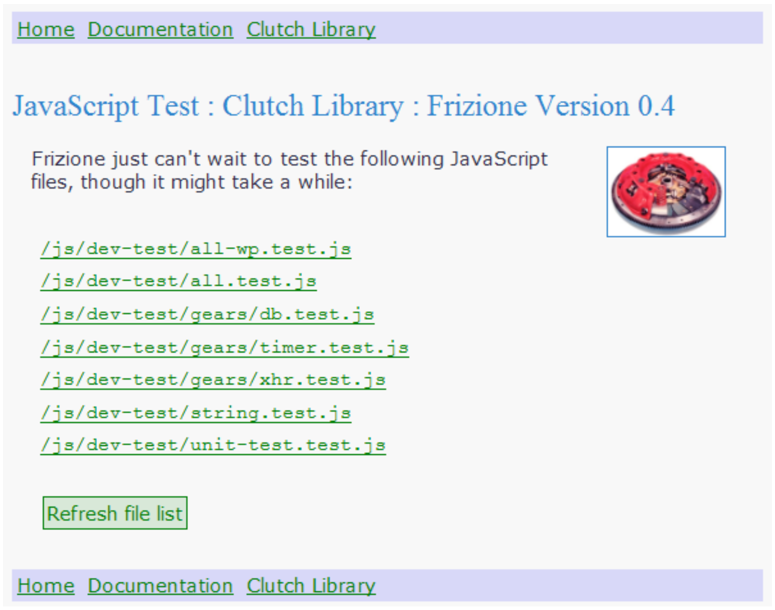
Successful minify

Frizione is delighted to inform you that the minify was completed with the following results:

```
  to: /js/browser-unittester.cjs
        size: 44960 characters
        minified size: 14323 characters
gzip: /js/browser-unittester.cjs.gz
        size: 4058 bytes
```

Home Documentation Clutch Library JavaScript Minify

The Test Page


Home Documentation Clutch Library

JavaScript Test : Clutch Library : Frizione Version 0.4

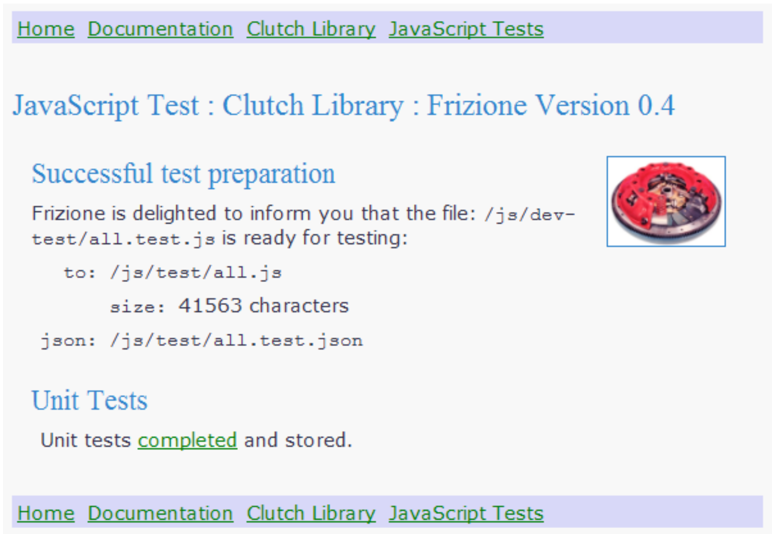
Frizione just can't wait to test the following JavaScript files, though it might take a while:

```
/js/dev-test/all-wp.test.js
/js/dev-test/all.test.js
/js/dev-test/gears/db.test.js
/js/dev-test/gears/timer.test.js
/js/dev-test/gears/xhr.test.js
/js/dev-test/string.test.js
/js/dev-test/unit-test.test.js
```

Refresh file list

Home Documentation Clutch Library

The main Test page provides links to each unit test file. By now it's clear that Clutch caches project file information, so if you make changes, don't forget to click on the refresh button. Clicking on any of the displayed links executes the unit tests, producing a Unit Test results page:


Home Documentation Clutch Library JavaScript Tests

JavaScript Test : Clutch Library : Frizione Version 0.4

Successful test preparation

Frizione is delighted to inform you that the file: /js/dev-test/all.test.js is ready for testing:

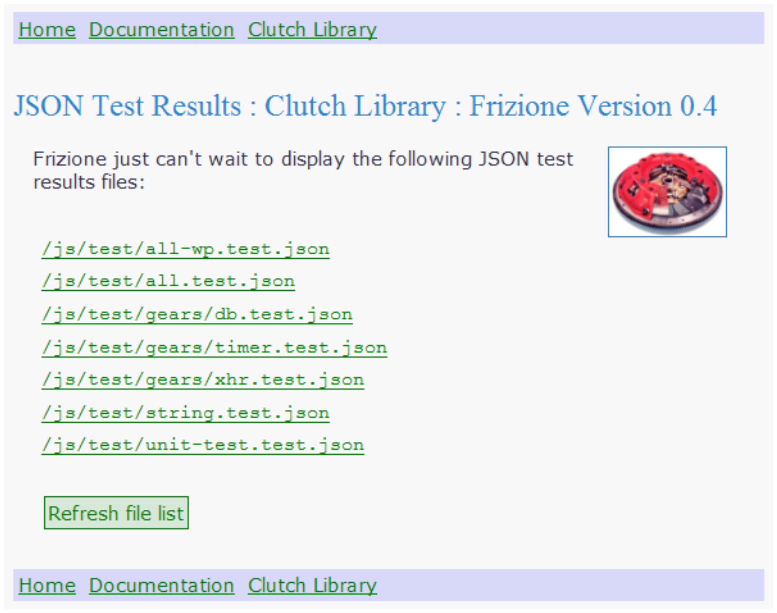
```
   to: /js/test/all.js
         size: 41563 characters
 json: /js/test/all.test.json
```

Unit Tests

Unit tests completed and stored.

Home Documentation Clutch Library JavaScript Tests

The Test Results Page


Home Documentation Clutch Library

JSON Test Results : Clutch Library : Frizione Version 0.4

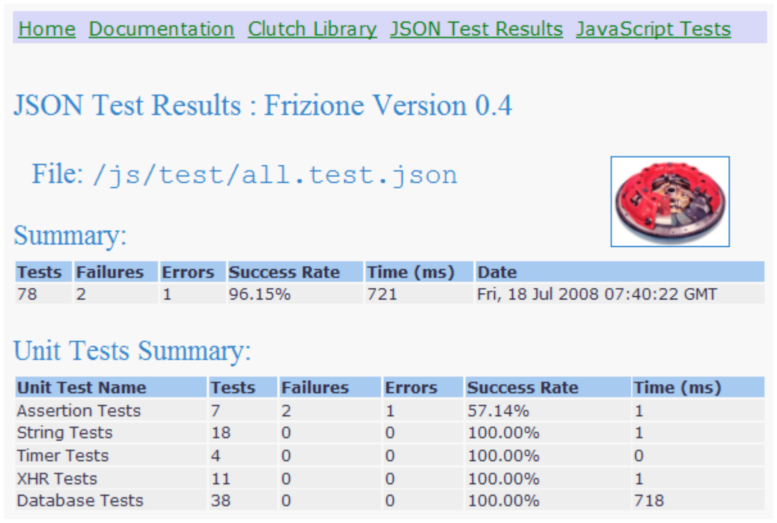
Frizione just can't wait to display the following JSON test results files:

/js/test/all-wp.test.json
/js/test/all.test.json
/js/test/gears/db.test.json
/js/test/gears/timer.test.json
/js/test/gears/xhr.test.json
/js/test/string.test.json
/js/test/unit-test.test.json

Refresh file list

Home Documentation Clutch Library

The main Test Results page provides links to each unit test results file. By now you're well aware that Clutch caches project file information, so if you make changes, don't forget to click on the refresh button. Clicking on any of the displayed links displays the unit test results:


Home Documentation Clutch Library JSON Test Results JavaScript Tests

JSON Test Results : Frizione Version 0.4

File: /js/test/all.test.json

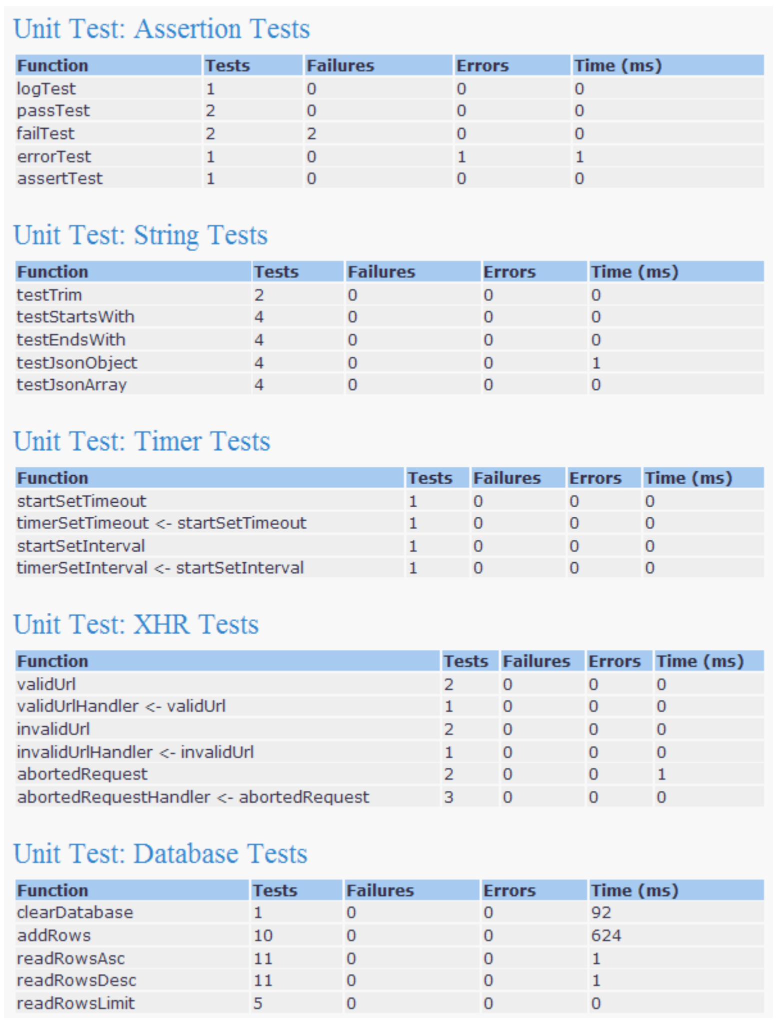
Summary:

Tests	Failures	Errors	Success Rate	Time (ms)	Date
78	2	1	96.15%	721	Fri, 18 Jul 2008 07:40:22 GMT

Unit Tests Summary:

Unit Test Name	Tests	Failures	Errors	Success Rate	Time (ms)
Assertion Tests	7	2	1	57.14%	1
String Tests	18	0	0	100.00%	1
Timer Tests	4	0	0	100.00%	0
XHR Tests	11	0	0	100.00%	1
Database Tests	38	0	0	100.00%	718

The unit test report starts with a general summary of the results.



Unit Test: Assertion Tests

Function	Tests	Failures	Errors	Time (ms)
logTest	1	0	0	0
passTest	2	0	0	0
failTest	2	2	0	0
errorTest	1	0	1	1
assertTest	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	1
testJsonArray	4	0	0	0

Unit Test: Timer Tests

Function	Tests	Failures	Errors	Time (ms)
startSetTimeout	1	0	0	0
timerSetTimeout <- startSetTimeout	1	0	0	0
startSetInterval	1	0	0	0
timerSetInterval <- startSetInterval	1	0	0	0

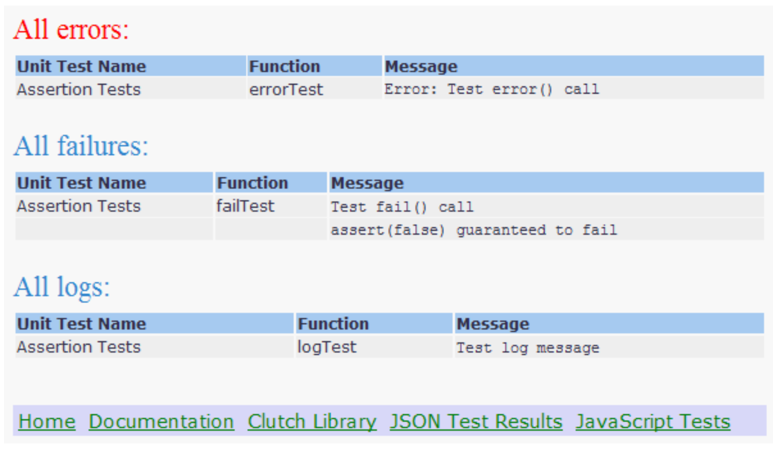
Unit Test: XHR Tests

Function	Tests	Failures	Errors	Time (ms)
validUrl	2	0	0	0
validUrlHandler <- validUrl	1	0	0	0
invalidUrl	2	0	0	0
invalidUrlHandler <- invalidUrl	1	0	0	0
abortedRequest	2	0	0	1
abortedRequestHandler <- abortedRequest	3	0	0	0

Unit Test: Database Tests

Function	Tests	Failures	Errors	Time (ms)
clearDatabase	1	0	0	92
addRows	10	0	0	624
readRowsAsc	11	0	0	1
readRowsDesc	11	0	0	1
readRowsLimit	5	0	0	0

This is followed by a more detailed report for each unit test.



All errors:

Unit Test Name	Function	Message
Assertion Tests	errorTest	`Error: Test error() call`

All failures:

Unit Test Name	Function	Message
Assertion Tests	failTest	`Test fail() call`
		`assert(false) guaranteed to fail`

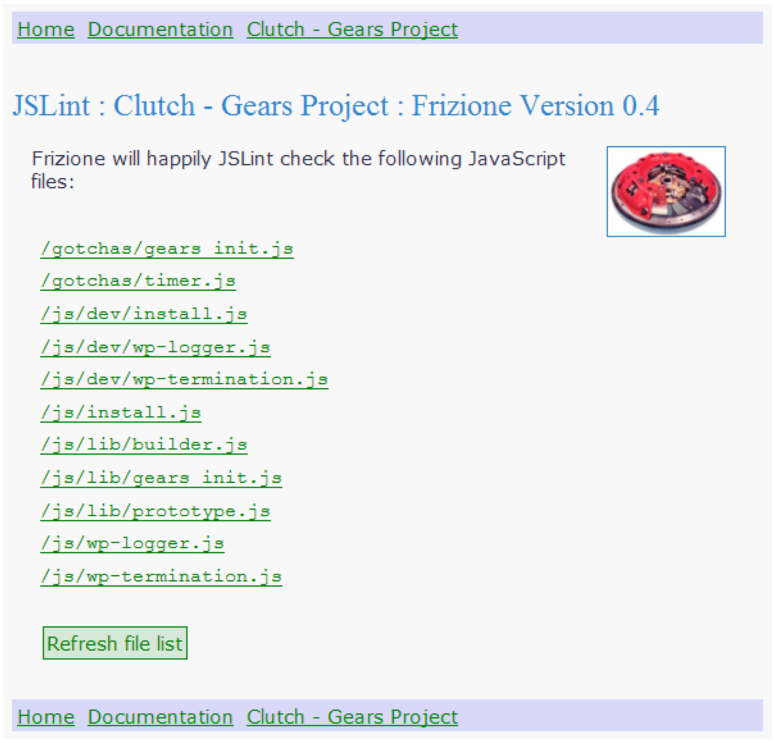
All logs:

Unit Test Name	Function	Message
Assertion Tests	logTest	`Test log message`

Home Documentation Clutch Library JSON Test Results JavaScript Tests

Finally, errors, failures and log messages are displayed.

The JSLint Page


Home Documentation Clutch - Gears Project

JSLint : Clutch - Gears Project : Frizione Version 0.4

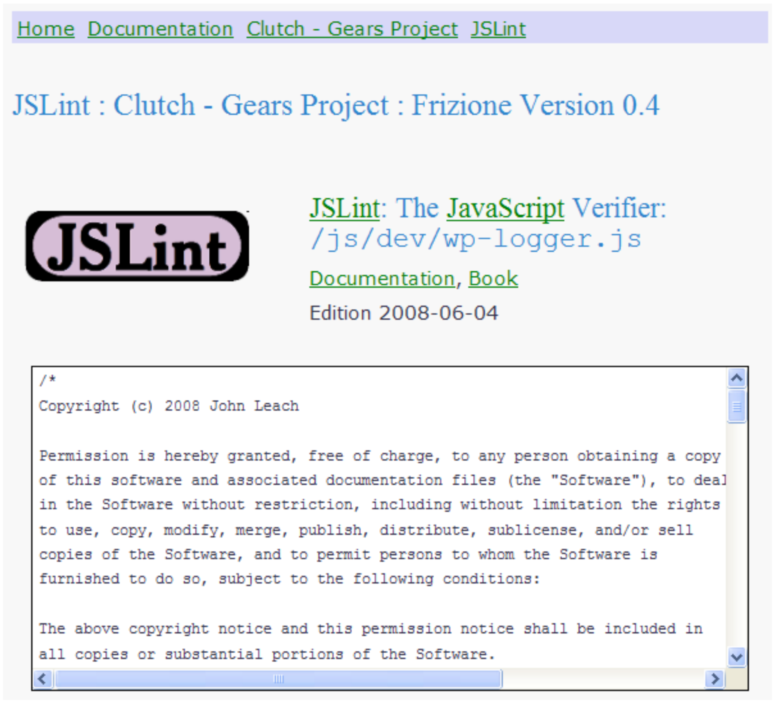
Frizione will happily JSLint check the following JavaScript files:

```
/gotchas/gears_init.js
/gotchas/timer.js
/js/dev/install.js
/js/dev/wp-logger.js
/js/dev/wp-termination.js
/js/install.js
/js/lib/builder.js
/js/lib/gears_init.js
/js/lib/prototype.js
/js/wp-logger.js
/js/wp-termination.js
```

Refresh file list

Home Documentation Clutch - Gears Project

The main JSLint page provides links to all JavaScript files which are not used for Join, Minify or Tests. Here too, Clutch caches project file information, so if you make changes, don't forget to click on the refresh button. Clicking on any of the displayed links executes JSLint on that file:


Home Documentation Clutch - Gears Project JSLint

JSLint : Clutch - Gears Project : Frizione Version 0.4

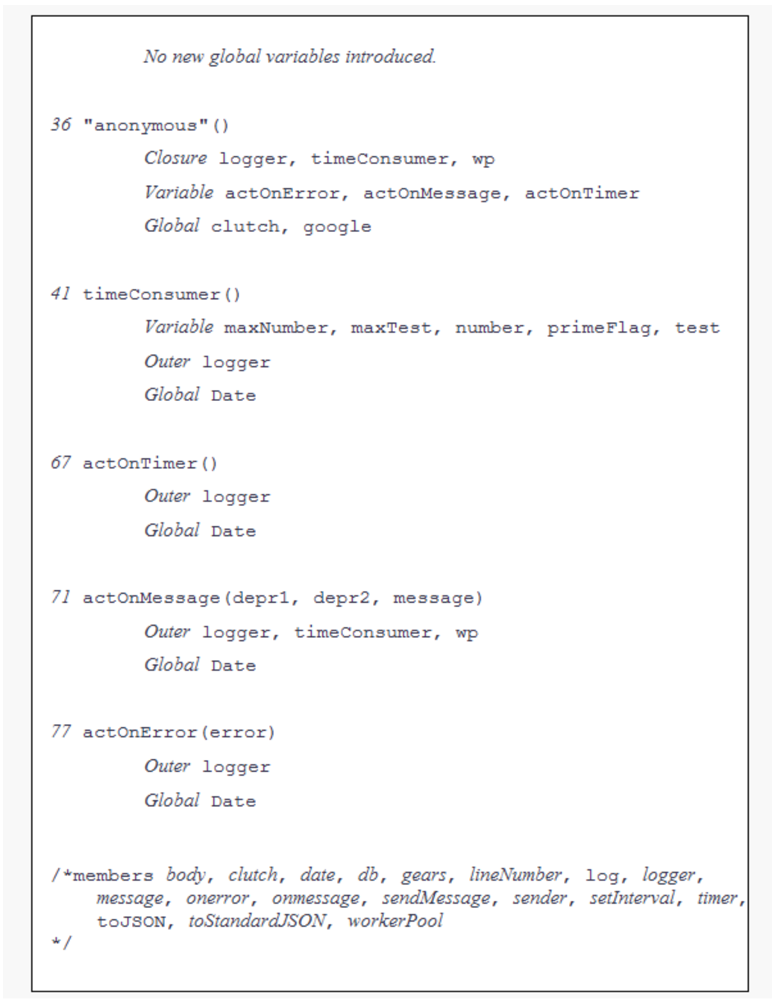
JSLint: The JavaScript Verifier: /js/dev/wp-logger.js

Documentation, Book

Edition 2008-06-04

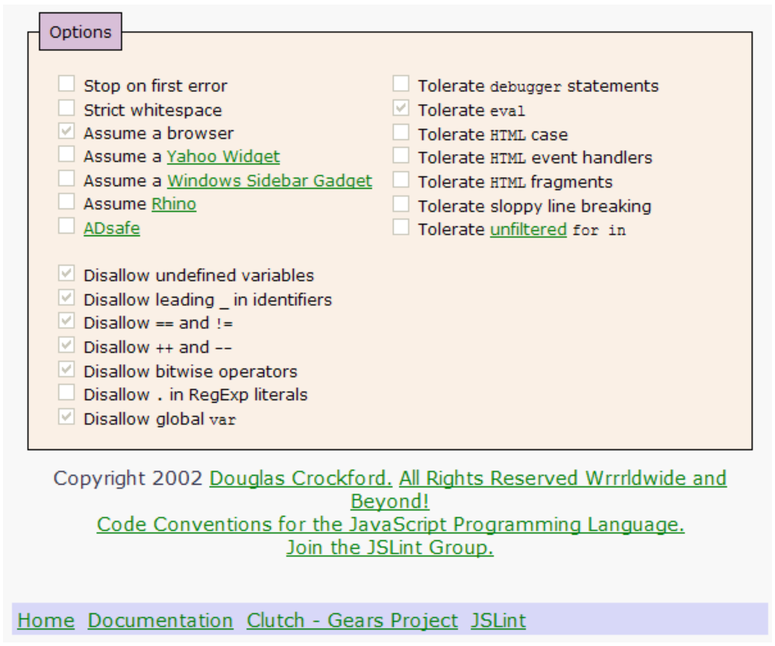
```
/*
Copyright (c) 2008 John Leach

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
```

First, the JavaScript file contents are displayed, and so can be used to check modifications 'on the fly'. You may need to refresh the page (usually F5).



```
        No new global variables introduced.

36 "anonymous"()
        Closure logger, timeConsumer, wp
        Variable actOnError, actOnMessage, actOnTimer
        Global clutch, google

41 timeConsumer()
        Variable maxNumber, maxTest, number, primeFlag, test
        Outer logger
        Global Date

67 actOnTimer()
        Outer logger
        Global Date

71 actOnMessage(depr1, depr2, message)
        Outer logger, timeConsumer, wp
        Global Date

77 actOnError(error)
        Outer logger
        Global Date

/*members body, clutch, date, db, gears, lineNumber, log, logger,
    message, onerror, onmessage, sendMessage, sender, setInterval, timer,
    toJSON, toStandardJSON, workerPool
*/
```

Then follows the JSLiint report. Error messages are shown in a shocking pink, probably to encourage you to correct them.



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`

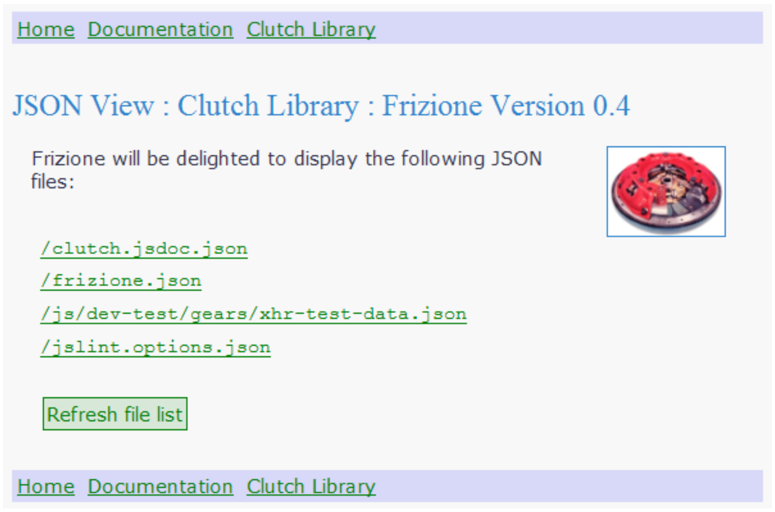
- ✓ Disallow undefined variables
- ✓ Disallow leading _ in identifiers
- ✓ Disallow == and !=
- ✓ Disallow ++ and --
- ✓ Disallow bitwise operators
- ☐ Disallow . in RegExp literals
- ✓ Disallow global `var`

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

Home Documentation Clutch - Gears Project JSLint

Finally the options used are displayed.

The JSON View Page



Home Documentation Clutch Library

JSON View : Clutch Library : Frizione Version 0.4

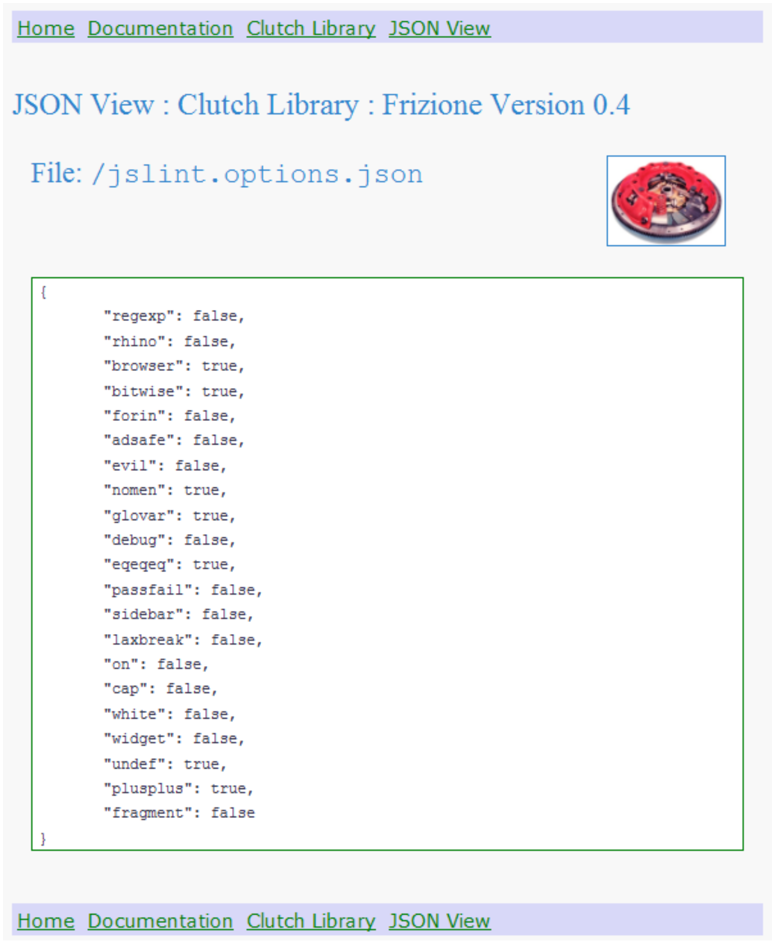
Frizione will be delighted to display the following JSON files:

/clutch.jsdoc.json
/frizione.json
/js/dev-test/gears/xhr-test-data.json
/jslint.options.json

Refresh file list

Home Documentation Clutch Library

The main JSON View page provides links to all JSON files which are not used for unit test results. Once again, Clutch caches project file information, so if you make changes, don't forget to click on the refresh button. Clicking on any of the displayed links displays the contents of the JSON file:



Home Documentation Clutch Library JSON View

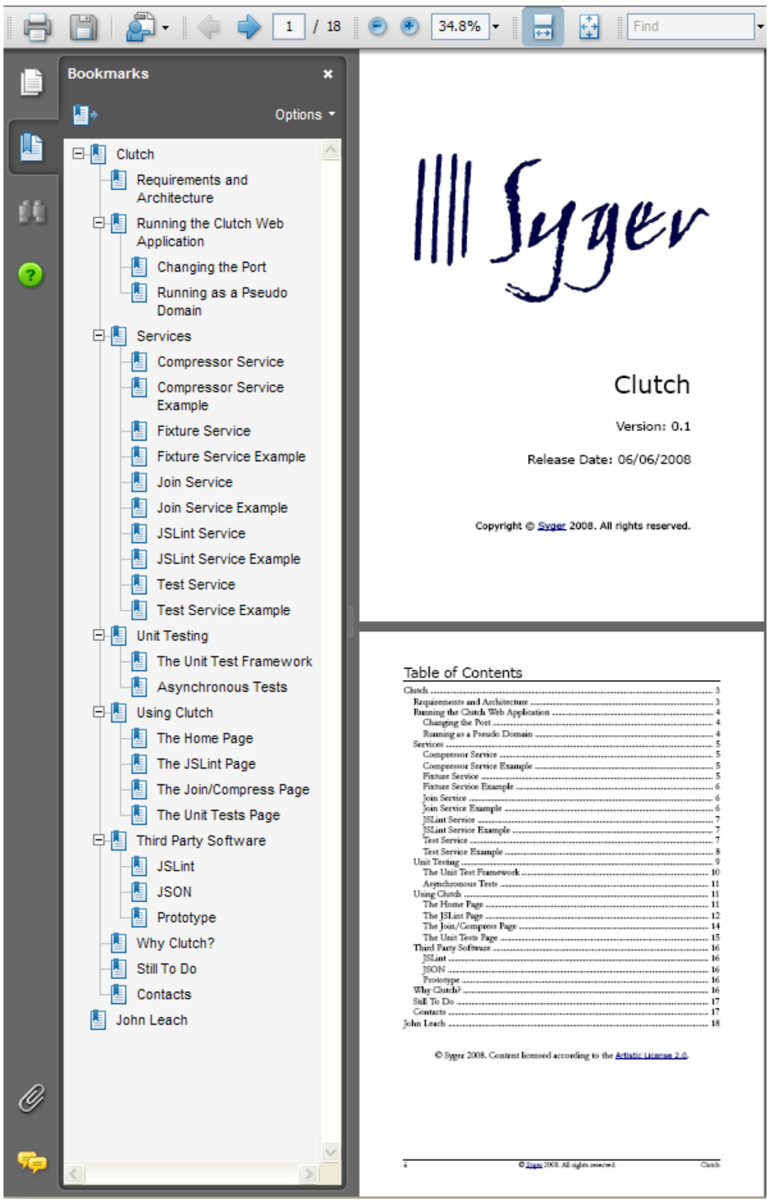
JSON View : Clutch Library : Frizione Version 0.4

File: /jslint.options.json

```
{
        "regexp": false,
        "rhino": false,
        "browser": true,
        "bitwise": true,
        "forin": false,
        "adsafe": false,
        "evil": false,
        "nomen": true,
        "glovar": true,
        "debug": false,
        "eqeqeq": true,
        "passfail": false,
        "sidebar": false,
        "laxbreak": false,
        "on": false,
        "cap": false,
        "white": false,
        "widget": false,
        "undef": true,
        "plusplus": true,
        "fragment": false
}
```

Home Documentation Clutch Library JSON View

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

Clutch

Version: 0.1

Release Date: 06/06/2008

Copyright © Syger 2008. All rights reserved.

Table of Contents

© 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. The source file locations all refer to `/Frizione-0.4/Frizione/frizione-projects/clutch/`.

Browser

Source: `js/dev/browser.js`

Namespace: `clutch.browser`

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

Introspect

Source: `js/dev/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: `js/dev/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: `js/dev/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: `js/dev/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: `js/dev/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: `js/dev/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 `js/dev/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 `js/dev/gears/wp-unit-test.js` for an example.

Database Utilities

Source: `js/dev/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: `js/dev/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

The latest release (0.4) is a great improvement in simplification and ease of use. The time taken in translating the code from Ruby to JavaScript was well worth it. It just seems to feel right now, and it certainly makes sense to have a JavaScript toll written in, er, JavaScript.

The following notes 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 simple file naming convention is just great. The tool has become much easier to use, I only have to write a couple of lines of comment commands inside the actual CSS or JavaScript file, so I think I've finally got a tool that I'll make great use of.

The So-So Parts

There are still a couple of ugly hacks in the code. I really want to find some way of amalgamating Java code which uses Rhino into one single jar. That probably means taking the Rhino sources, and merging Helma, YUICompressor, ShrinkSafe, and other Rhino based code into one project. But I'll need help on that, so I'm going to see if this can be a collaborative effort with the various authors of those fine programs.

Now that I've got a 'true' JavaScript tool written in JavaScript, there are other tools which I'd like to amalgamate, such as JsDoc Toolkit, to name but one.

The Bad Parts

Fortunately, in the latest release, most of the bad parts have disappeared.

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.

Clutch Development Notes

I've spent several days checking out various web application servers for the next release of Clutch. The reason for this is that I'd like to use more JavaScript tools written in JavaScript, such as JSDoc Toolkit, and unfortunately, Ruby doesn't really have a very good wrapper to a JavaScript engine such as SpiderMonkey. There is a project, but it seems to be comatose, and there is also a new project called Johnson, but this is still in the early stages of development.

Web Application Servers

Since it seems a reasonable idea to move the server code into a server side JavaScript environment, the big question is, which web application server do I choose? There are plenty to choose from, mostly Apache/SpiderMonkey or Java/Rhino based, but I reduced the list to three, mostly based on my own home brew 'tyre kicking' process – well we all have one don't we?

The three finalists were:

- Aptana Jaxer, an Apache/SpiderMonkey combination,
- Phobos, a Java/Rhino combination,
- Helma, a Java/Rhino combination.

Although I had some doubts as to its activity level, my first choice was Phobos – until I found that there is no standalone download, you have to download Netbeans (over 100 MB) which contains a module for Phobos. Aptana Jaxer is interesting, but I'm not too happy with a page based controller pattern, nor with the fact that it's almost impossible to extend with C or Java libraries. Which left me with the mature and stable Java based Helma, which was only slightly behind Phobos on my points system - Phobos theoretically handles multiple scripting languages, whereas Helma is JavaScript only.

Helma was a little difficult to understand in terms of its philosophy, however after a few days I think I managed to master it, and the results fell good, especially the low quantity of code. I said quantity.

Similar Projects

This list provides links to projects that I consider to be similar to Clutch. I have only had time to read the advertising, so you'll have to make your own judgement (as always) of the software.

jsLex: this is a profiling tool, which integrates with Aptana Studio, or Eclipse. Designed to help produce efficient JavaScript and CSS, it also provides file concatenation and size reduction.

Maven JSTools Plugin: a Maven interface to a set of JavaScript reporting and documentation tools like JSDoc Toolkit and JSLint, along a simple approach for building JS artifacts and use them as dependencies in your Maven-based projects

newjs: a quite similar concept to Clutch, using the command line, and the Prototype unit testing framework. It provides file concatenation, and uploading to a web site.

JavaScript Libraries of Note

There are an enormous number of JavaScript projects lingering in various open source project sites on the Internet. The majority are either content free, incomplete, or abandoned. Those which pass my 'tyre kicking' process, and seem to have made enough effort to be worth further investigation are listed below. Again, I have only had time to read the advertising, so you'll have to make your own judgement (as always) of the software.

enrichmentkit: JavaScript lib for RIAs featuring URL state representation, rewriting, dispatching, named history entries, model locating, and object observance.

fbug: the Firebug Firefox add-on project.

JavaScriptMVC: a framework that brings methods to the madness of JavaScript development (their words, not mine).

jslibs: standalone Javascript development environment with general purpose native libraries.

jsSHA: a JavaScript implementation of the entire family of SHA hashes as defined in FIPS 180-2 (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512).

packer: Dean Edwards' JavaScript compressor.

trimpath: open-source web technologies, focused around JavaScript, synchronization, MVC, occasionally connected computing, and Google Gears. The Next Action offline web application is particularly interesting.

simile-widgets: A toolbox of several web widgets and APIs originated from the MIT Simile project. The timeline example is particularly fascinating.

Other Interesting Links

Higher Order JavaScript: a comparison of JavaScript with Perl, using the book Higher Order Perl as the example.

Ajax (and More) Frameworks

This is not an exhaustive list, that would probably require a book. Just some of the better known frameworks, most of which you'll already know about – but then again, you might find a surprise or two.

dojo: the Dojo Toolkit.

jMaki: a client-server framework for creating Jaax applications and mashups.

jQuery: is designed to change the way that you write JavaScript (their words, not mine).

MochiKit: makes JavaScript suck less (their words, not mine).

MooTools: a compact, modular, Object-Oriented JavaScript framework designed for the intermediate to advanced JavaScript developer.

Prototype: a JavaScript Framework that aims to ease development of dynamic web applications. Normally used together with script.acul.ous: provides you with easy-to-use, cross-browser user interface JavaScript libraries to make your web sites and web applications fly (their words, not mine).

qooxdoo: a comprehensive and innovative Ajax application framework.

YUI: the Yahoo! User Interface Library.

Frizione – Clutch Version History

by John Leach

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

Version 0.4 – 16/07/2008

This is a major shift both in the technology used, and, hopefully, in ease of use. The web application has been rewritten in JavaScript (with a heavy Java accent), using the Helma web application server. Most services are now available using a simple file naming convention, and commands are stored in the files on which the commands are performed. Much less typing, no HTML forms at all.

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.