
Documentación CasperJs

1.1.0 Release-DEV

Nicolas Perriault

Aug 23 de, 2017

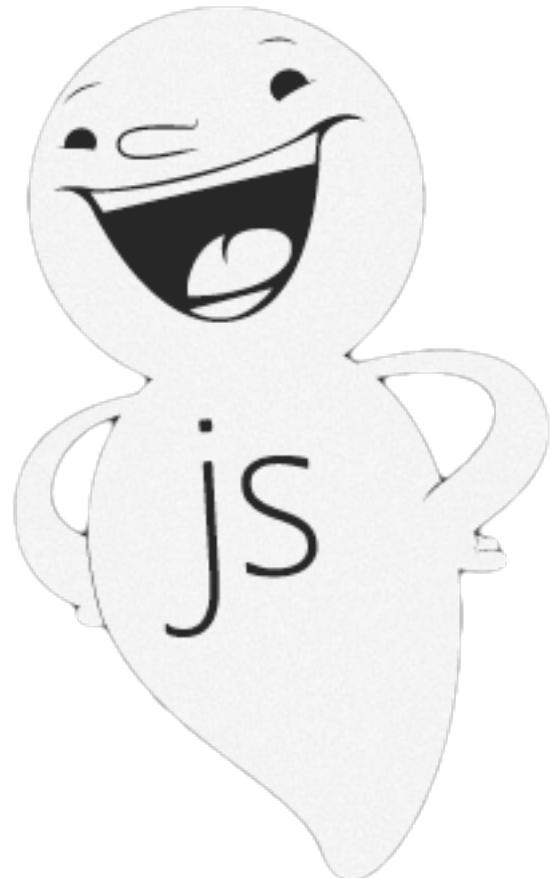
Contenido

1 Instalación	3
1.1 Requisitos previos	3
1.2 Instalación del Homebrew (OSX)	4
1.3 Instalación desde la NGP	4
1.4 Instalación desde git	4
1.5 Instalación desde un archivo	5
1.6 CasperJS en Windows	5
1.7 Errores conocidos y limitaciones	6
2 Inicio rápido	7
2.1 Un script mínimo raspado	7
2.2 Ahora vamos a raspar Google!	8
2.3 versión CoffeeScript	9
2.4 Un script de pruebas mínimas	10
3 Uso de la línea de comandos	11
3.1 <i>casperjs</i> <td>12</td>	12
3.2 valores de los parámetros sin procesar	13
4 selectores	15
4.1 CSS3	15
4.2 XPath	dieciséis
5 Prueba	17
5.1 Unidad de prueba	17
5.2 Pruebas de navegador	18
5.3 Configuración de las opciones de Casper en el entorno de prueba	20
5.4 Las técnicas avanzadas	20
5.5 argumentos y opciones de comandos de prueba	20
5.6 Exportación de los resultados en formato XUnit	21
5.7 CasperJS pruebas propias	22
5.8 Extensión de Casper para Pruebas	23
Documentación 6 API	25
6.1 La Casper módulo	25
6.2 La clientutils módulo	70
6.3 La colorizer módulo	77

6.4 La ratón módulo	80
6.5 La ensayador módulo	82
6.6 La utils módulo	99
7 módulos de escritura CasperJS	103
8 Eventos y Filtros	105
8.1 Emisión de eventos es el propietario	105
8.2 Extracción de eventos	106
8.3 Eventos referencia	107
8.4 filtros Referencia	119
9 registro	123
10 que se extiende	125
10.1 El uso de CoffeeScript	127
11 Depuración	129
11.1 Utilice el modo detallado	129
11.2 Hook en los eventos utilizando profunda	130
11.3 volcado valores serializado a la consola	130
11.4 Localizar el mismo en los módulos	130
11.5 Nombre sus cierres	130
12 Preguntas	133
12.1 Es CasperJS una biblioteca Node.js?	134
12.2 estoy atascado! Creo que hay un error! ¿Que puedo hacer?	134
El 12.3 casper.test propiedad es inde fi nida, no puedo escribir cualquier prueba!	134
12.4 sigo copiar y pegar cosas en mis scripts de prueba, eso es aburrido	134
12.5 ¿Cuál es la política de control de versiones de CasperJS?	134
12.6 ¿Puedo usar jQuery con CasperJS?	135
12.7 ¿Puedo usar CasperJS sin utilizar el casperjs ¿ejecutable?	135
12.8 ¿Cómo puedo coger HTTP 404 y otra de estado de los códigos?	136
12.9 De dónde viene CasperJS escribir su registro fi l?	136
12.10 ¿Qué es esto misteriosa __ utils__ ¿objeto?	136
12.11 ¿Cómo funciona entonces() y trabajar la pila paso?	137
12.12 Estoy teniendo tiempos difíciles de descargar archivos utilizando descargar()	138
12.13 ¿Es posible conseguir una navegación paralelo usando CasperJS?	138
12.14 ¿Puedo acceder y manipular elementos DOM directamente desde el entorno CasperJS?	138
12.15 ¿Por qué no puedo crear una nueva Casper ejemplo, en un entorno de prueba?	139
12.16 Muy bien, honestamente, estoy atascado con Javascript.	139
12.17 ¿Cómo se utiliza la página PhantomJS módulo de API en casperjs?	139
12.18 ¿Cómo proporciono mi implementación de un recurso remoto?	140
12.19 Me estoy fallo de la prueba intermitente, lo que puedo hacer para fi jar ellos?	140
13 Cookbook	141
13.1 Creación de un servicio web	141
13.2 de secuencias de comandos para comprobar automáticamente una página de errores 404 y 500	142
13.3 drag & drop Prueba	144
13.4 Paso de parámetros en sus pruebas	144
14 Cambios	147
15 Actualización	149
15.1 La actualización a 1.1	149

16 Problemas conocidos	153
16.1 PhantomJS	153
17 créditos	155
17.1 Autor	155
17.2 Colaboradores	155
17.3 Logo	156
18 Licencia	159
19 Comunidad	161

CasperJS es un navegador de scripting y sin cabeza para navegadores, escrito en PhantomJS (WebKit) y SlimerJS (Geco).



CAPÍTULO 1

Instalación

CasperJS se pueden instalar en Mac OS X, Windows y la mayoría de Linuxes.

Requisitos previos

- PhantomJS 1.9.1 o mayor. Por favor leer el [instrucciones de instalación para PhantomJS](#)
- Pitón 2.6 o mayor para casperjs en el compartimiento/ directorio

Nota: CoffeeScript no se admite de forma nativa en versiones PhantomJS 2.0.0 y anteriores. Si se va a utilizar café- escritura que tendrá que transpile en la vainilla Javascript. Ver [Problemas conocidos](#) para más detalles.

Nuevo en la versión 1.1.

- Experimental: como de 1.1.0-beta1, SlimerJS 0.8 o superior para ejecutar las pruebas en contra del Gecko (Firefox) en lugar de Webkit (sólo añadir `-motor = slimerjs` a las opciones de línea de comandos). Los desarrolladores SlimerJS documentados [la compatibilidad API PhantomJS de SlimerJS](#) tanto como [las diferencias entre PhantomJS y SlimerJS](#). Tenga en cuenta que se sabe que se rompe de apoyo coffeeescript como de SlimerJS 0.9.6; estamos investigando esta cuestión.

Nuevo en la versión 1.1.0-beta4.

Advertencia: Versiones antes de 1.1.0-beta4 que fueron instalados a través de NPM requieren un `fi` no específico versión PhantomJS por medio de una dependencia MNP. Esto dio lugar a mucha confusión y problemas contra CasperJS no funciona correctamente si se instala a través de la NGP. A partir de la instalación de un motor (PhantomJS, SlimerJS) será una verdadera condición, independientemente del método de instalación que se elija para CasperJS.

Instalación del Homebrew (OSX)

La instalación de ambos PhantomJS y CasperJS se puede lograr utilizando [homebrew](#), Un gestor de paquetes populares para Mac OS X.

Por encima de todo, no se olvide de actualizar formulas:

```
actualización de $ brebaje
```

. Para la versión 1.1 * (recomendado):

```
$ Brebaje instalar casperjs
```

Si ya ha instalado casperjs y quieren tener la última versión (estable | devel), el uso mejorará:

```
$ casperjs actualización de cerveza
```

Actualizar sólo la actualización a la última rama de lanzamiento (1.0.x | 1.1.0-dev).

Instalación desde la NGP

Nuevo en la versión 1.1.0-beta3. Puede instalar

usando CasperJS [NPM](#):

- Para la mayoría de los usuarios (la actual versión 1.1.0-Beta4):

```
$ NPM instalar casperjs -g
```
- Si desea una especificidad de versión anterior:
 - Para beta3:

```
$ NPM instalar -g casperjs@1.1.0-beta3
```
 - Para beta2:

```
$ NPM instalar -g casperjs@1.1.0-beta2
```
- Si desea instalar el maestro actual de git utilizando NPM:

```
$ NPM instalar git -g + https://github.com/casperjs/casperjs.git
```

Nota: Los - g y -g hace que el casperjs ejecutable disponible en todo el sistema.

Advertencia: Mientras CasperJS se puede instalar a través de la NGP, *no es un módulo NodeJS* y no funcionará con NodeJS fuera de la caja. No se puede cargar mediante el uso de Casper require () en casperjs nodo. Tenga en cuenta que CasperJS no es capaz de utilizar una gran mayoría de los módulos de NodeJS por ahí. Experimentar y utilizar su mejor juicio.

Instalación desde git

La instalación se puede lograr utilizando [Git](#). El código está alojado principalmente en [Github](#).

De la rama principal

```
$ Git clone git: $ cd //github.com/casperjs/casperjs.git casperjs  
$ Ln -sf `pwd` / bin / casperjs / usr / local / bin / casperjs
```

Una vez PhantomJS y CasperJS instalado en su máquina, debe obtener algo como esto:

```
$ PhantomJS --version  
1.9.2  
$ casperjs  
CasperJS versión 1.1.0-beta4 en / Users / Niko / sites / casperjs, utilizando PhantomJS versión 1.  
.. 9.2  
# . . .
```

O si SlimerJS es lo suyo:

```
$ Slimerjs --version  
Innophi SlimerJS 0.8pre, Derechos de Autor 2012-2013 Laurent Jouanneau y Innophi $ casperjs  
  
CasperJS versión 1.1.0 en / Users / Niko / sites / casperjs, utilizando slimerjs versión 0.8.0
```

Ahora está listo para escribir su [primer guión fi](#)!

Instalación desde un archivo

Puede descargar archivos con etiquetas de código de CasperJS:

Última versión de desarrollo (rama principal):

- <https://github.com/casperjs/casperjs/zipball/master> (cremallera)
- <https://github.com/casperjs/casperjs/tarball/master> (Tar.gz)

La última versión estable:

- <https://github.com/casperjs/casperjs/zipball/1.1.0> (cremallera)
- <https://github.com/casperjs/casperjs/tarball/1.1.0> (Tar.gz) Las operaciones son

entonces el mismo que con un git checkout.

CasperJS en Windows

PhantomJS adiciones de instalación

- Adjuntar "; C: \ PhantomJS" para usted CAMINO Variable ambiental.
- Modificar esta ruta apropiada si ha instalado PhantomJS a una ubicación diferente.

Casperjs adiciones de instalación

Nuevo en la versión 1.1.0-beta3.

- Adjuntar "; C:\casperjs\bin" para usted CAMINO variable de entorno (para las versiones antes append 1.1.0-beta3 "; C:\casperjs\batchbin" para usted CAMINO Variable ambiental).
- Modificar esta ruta apropiada si ha instalado CasperJS a una ubicación diferente.
- Si el equipo utiliza tanto gráficos discretos e integrados, es necesario deshabilitar selección automática y seleccionar de forma explícita procesador de gráficos - de lo contrario salida() no va a salir de Casper. Ahora puede ejecutar cualquier script Casper regulares de esa manera:

```
C:> casperjs MyScript.js
```

salida tintadas

Nota: Nuevo en la versión 1.1.0-beta1. Los usuarios de Windows obtener una salida coloreada si `ansicon` está instalado o si el usuario está utilizando `ConEmu` con colores ANSI habilitado.

Compilación (Opcionalmente)

- .NET Framework 3.5 o mayor (o Mono 2.10.8 o superior) para `casperjs.exe` en el compartimiento/ directorio

Errores conocidos y limitaciones

- Debido a su naturaleza asíncrona, CasperJS no funciona bien con `REPL PhantomJS'`.

CAPÍTULO 2

Inicio rápido

Una vez CasperJS es *correctamente instalado*, se puede escribir el guión primero. Se puede utilizar sin formato *Javascript* (o *CoffeeScript* con versiones PhantomJS antes 2.0).

Insinuación: Si no estás muy cómodo con Javascript, una *Preguntas entrada dedicada* te está esperando.

Una secuencia de comandos mínima raspado

Lanzar su editor favorito, crear y guardar una sample.js así:

```
var Casper = exigir('Casper').crear();

casper.start ('Http://casperjs.org', función () {
    esta . eco( esta . getTitle ());
});

casper.thenOpen ('Http://phantomjs.org', función () {
    esta . eco( esta . getTitle ());
});

casper.run ();
```

Ejecutarlo (con Python):

```
$ Casperjs sample.js
```

Ejecutarlo (con nodo): más información aquí - <https://github.com/casperjs/casperjs/issues/1864>

```
$ Nodo casperjs.js sample.js
```

Ejecutarlo (con PhantomJS):

```
$ PhantomJS casperjs.js sample.js
```

Ejecutarlo (en Windows):

```
C:\casperjs\bin> casperjs.exe sample.js
```

Usted debe obtener algo como esto:

```
$ Casperjs sample.js
CasperJS, un script de navegación y utilidad para probar PhantomJS PhantomJS: WebKit sin cabeza con la API de
JavaScript
```

Lo qué acabamos de hacer?

1. creamos una nueva *Casper* ejemplo
2. empezamos y abrimos <http://casperjs.org/>
3. *una vez* la página se ha cargado, nos preguntamos para imprimir el título de esa página web (el contenido de su <title> etiqueta)
4. *entonces* abrimos otra url, <http://phantomjs.org/>
5. *una vez* la nueva página se ha cargado, nos preguntamos para imprimir su título demasiado
6. ejecutamos todo el proceso

Ahora vamos a raspar Google!

En el siguiente ejemplo, vamos a consulta en Google por dos períodos consecutivos, “*casperjs*” y “*PhantomJS*”, agregar los enlaces de resultado en una norma Formación y enviar el resultado a la consola. Lanzar su editor favorito y guardar el código Javascript a continuación en una googlelinks.js fi l:

```
var campo de golf = [];
var Casper = exigir( 'Casper' ).crear();

función getLinks () {
    var campo de golf = documento .querySelectorAll ( 'H3.ra' );
    regreso Formación .prototype.map.call (enlaces, función ( e ) {
        regreso e.getAttribute ( 'Href' );
   }); }

casper.start ( 'Http://google.fr' , función () {
    // Espera a que la página se carga
    esta .waitForSelector ( 'Forma [action = "/ búsqueda"]' );
});

casper.then ( función () {
    // búsqueda de " casperjs de forma Google
    esta . llenar( 'Forma [action = "/ búsqueda"]' , {Q : " casperjs } , cierto );
});

casper.then ( función () {
    // resultados agregados para la búsqueda de los " casperjs
```

```

campo de golf = esta . evaluar (getLinks);
// ahora buscar "PhantomJS rellenando el formulario de nuevo
esta . llenar( "Forma [action = "/ búsqueda"]" , {Q : " PhantomJS "}, cierto );
});

casper.then ( función () {
    // resultados agregados para la búsqueda de los " PhantomJS "
    campo de golf = links.concat ( esta . evaluar (getLinks)); });

casper.run ( función () {
    // resultados de eco de alguna manera bastante
    esta . echo (links.length + "Se han encontrado enlaces: ");
    esta . eco( '^' + links.join (' \n ')).salida();
});

```

Ejecutarlo:

```

$ casperjs googlelinks.js 20 enlaces
encontrados:
- https://github.com/casperjs/casperjs
- https://github.com/casperjs/casperjs/issues/2
- https://github.com/casperjs/casperjs/tree/master/samples
- https://github.com/casperjs/casperjs/commits/master/
- http://www.facebook.com/people/Casper-Js/100000337260665
- http://www.facebook.com/public/Casper-Js
- http://hashtags.org/tag/CasperJS/
- http://www.zerotohundred.com/newforums/members/casper-js.html
- http://www.yellowpages.com/casper-wy/js-enterprises
- http://local.trib.com/casper+wy/j+s+chinese+restaurant.zq.html
- http://www.phantomjs.org/
- http://code.google.com/p/phantomjs/
- http://code.google.com/p/phantomjs/wiki/QuickStart
- http://svay.com/blog/index/post/2011/08/31/Paris-JS-10-%3A-Introduction-%C3%A0-
.. PhantomJS
- https://github.com/ariya/phantomjs
- http://dailyjs.com/2011/01/28/phantoms/
- http://css.dzone.com/articles/phantom-js-alternative
- http://pilvee.com/blog/tag/phantom-js/
- http://ariya.blogspot.com/2011/01/phantomjs-minimalistic-headless-webkit.html
- http://www.readwriteweb.com/hack/2011/03/phantomjs-the-power-of-webkit.php

```

versión CoffeeScript

También puede escribir scripts Casper utilizando el sintaxis CoffeeScript :

```

getLinks = ->
  links = documento .querySelectorAll "H3.r un"
  Formación :: enlaces map.call, (E) -> e.getAttribute "Href"

  links = []
  Casper = exigir( 'Casper' ).crear()

  casper.start "Http://google.fr/" , ->
    # buscar " de forma casperjs Google

```

```
@llenar "Forma [action = '/ búsqueda']", q: "casperjs", cierto

casper.then ->
    # los resultados agregados para la búsqueda de los "casperjs"
    links = getLinks @evaluate
    # buscar "de forma PhantomJS Google"
    @llenar "Forma [action = '/ búsqueda']", q: "PhantomJS", cierto

casper.then ->
    # concat resultados para la búsqueda de los "PhantomJS"
    links = links.concat @evaluate (getLinks)

casper.run ->
    # mostrar los resultados
    links.length @echo + "Enlaces encontrados:"
    @echo( "-" + links.join ("\\N -")).salida()
```

Sólo recuerde que su fichero la secuencia de comandos con el. café extensión.

Nota: CoffeeScript no se admite de forma nativa en versiones PhantomJS 2.0.0 y anteriores. Si se va a utilizar café- escritura que tendrá que transpile en la vainilla Javascript. Ver [Problemas conocidos](#) para más detalles.

Un script de pruebas mínimas

CasperJS es también una [marco de pruebas](#); scripts de prueba son ligeramente diferentes de los de rascado, aunque comparten la mayor parte de la API.

Un script de prueba más simple:

```
// hello-Test.js
casper.test.begin( "Hola, prueba!", 1, función ( prueba ) {
    test.assert( cierto );
    test.done();
});
```

Ejecutarlo utilizando el prueba casperjs subcomando:

```
$ Casperjs la prueba del archivo de prueba
hola-Test.js: hello-Test.js
# Hola, prueba!
Asunto PASS es estrictamente cierto
PASAR 1 prueba ejecutada en 0.023s, 1 pasado, 0 incorrectos, 0 dudosa, 0 omitidos.
```

Nota: Como se puede ver, no hay necesidad de crear una Casper instancia en un script de prueba como una preconfigurada ya se ha puesto a disposición para usted.

Puede leer más sobre las pruebas en el [sección dedicada](#).

CAPÍTULO 3

Usando la línea de comandos

CasperJS barcos con un analizador integrado de línea de comandos en la parte superior del analizador PhantomJS, que se encuentra en el cli módulo. Expone argumentos pasados como los posicionales y opciones nombradas

UN Casper instancia siempre contiene un producto listo para usar cli propiedad para facilitar el acceso a estos parámetros, por lo que no tiene que preocuparse acerca de la manipulación de la cli módulo API de análisis. Vamos a considerar esta simple escritura de Casper:

```
var Casper = exigir("Casper").crear();

casper.echo ("Casper CLI pasó args:" ); exigir("utilidades") .dump
(casper.cli.args);

casper.echo ("Casper CLI pasó opciones:" ); exigir("utilidades") .dump
(casper.cli.options);

casper.exit();
```

Nota: Tenga en cuenta que los dos Casper-path y cli opciones; éstos se pasan a la escritura de Casper a través de la casperjs ejecutable de Python.

resultados de la ejecución:

```
$ Casperjs Test.js arg1 arg2 arg3 - foo = bar - plop anotherarg Casper CLI pasó args : [
    "Arg1",
    "Arg2",
    "Arg3",
    "Anotherarg"
]
Casper CLI pasó opciones : {
    "Casper-path" : "/ Users / Niko / sites / casperjs",
    "Cli" : cierto,
```

```
"Foo": "bar",
"plaf": cierto
}
```

Conseguir, la comprobación o dejar caer parámetros:

```
var Casper = exigir("Casper").crear(); casper.echo(casper.cli.has(0));
casper.echo(casper.cli.get(0)); casper.echo(casper.cli.has(3));
casper.echo(casper.cli.get(3)); casper.echo(casper.cli.has("Foo"));
casper.echo(casper.cli.get("Foo")); casper.cli.drop("Foo"); casper.echo(
(casper.cli.has("Foo"))); casper.echo(casper.cli.get("Foo")); casper.exit();
```

resultados de la ejecución:

```
$ Casperjs Test.js arg1 arg2 arg3 --foo = bar --plop anotherarg cierto arg1 verdadera
```

```
anotherarg
verdadera barra
falsa indefinido
```

Insinuación: Es posible que necesite ajustar la opción que contenga un espacio entre comillas dobles escapado de Windows. -foo = \"barra espaciadora\"

Insinuación: ¿Qué pasa si usted quiere comprobar si los hay arg u opción se ha pasado a la secuencia de comandos? Aquí tienes:

```
// extracción de opciones por defecto aprobadas por el ejecutable de Python
casper.cli.drop("Cli"); casper.cli.drop("Casper-path");

Si (casper.cli.args.length === 0 && Objeto.keys(casper.cli.options).length === 0) {Casper.echo("No hay ni arg opción de pasar").salida();}
```

***casperjs* opciones nativas**

Nuevo en la versión 1.1. los *casperjs* comando tiene tres opciones

disponibles:

- - - directo: para imprimir los mensajes de registro a la consola
- - - log-level = [depuración | información | advertencia | error] para establecer el *nivel de registro*

- - - motor = [PhantomJS | slimerjs] para seleccionar el motor del navegador que desea utilizar. CasperJS apoya PhantomJS (por defecto) que se ejecuta en Webkit, y SlimerJS que corre Gecko.

Advertencia: En desuso desde la versión 1.1. Los --directo opción se ha cambiado el nombre a - verboso. A pesar de que --directo seguirá funcionando, ahora se considera en desuso.

Ejemplo:

```
$ Casperjs --verbose --log-level = MyScript.js de depuración
```

Por último, pero no menos importante, todavía se puede utilizar todas las opciones estándar PhantomJS CLI como lo haría con cualquier otro script PhantomJS:

```
$ Casperjs --web-security = sin --galletas-file = / tmp / mycookies.txt MyScript.js
```

Insinuación: Para recordar lo que las opciones de la CLI PhantomJS nativos están disponibles, corren el PhantomJS --help mando. SlimerJS es compatible con casi las mismas opciones que PhantomJS.

valores de los parámetros primas

Nuevo en la versión 1.0.

Por defecto, el objeto cli procesará cada argumento pasado y las arrojó sobre el tipo detectado apropiada; script de ejemplo:

```
var Casper = exigir( 'Casper' ).crear();
var utils = exigir( 'utils' );

utils.dump (casper.cli.get ( 'Foo' ));

casper.exit ();
```

Si ejecuta esta secuencia de comandos:

```
$ Casperjs c.js --foo = 01234567 1234567
```

Como se puede ver, el 01234567 valor ha sido arrojado a una *Número*.

Si desea que la cadena original, utilice el crudo propiedad de la cli objeto, que contiene los valores en bruto de los parámetros pasados:

```
var Casper = exigir( 'Casper' ).crear();
var utils = exigir( 'utils' );

utils.dump (casper.cli.get ( 'Foo' )); utils.dump (casper.cli.raw.get ( 'Foo' ));

casper.exit ();
```

Ejemplo de uso:

```
$ Casperjs c.js --foo = 01234567 1234567 "01234567"
```

Para números muy largos, utilice el crudo la propiedad ya que hay una limitación de la escritura ECMA con una precisión numérica de hasta 17 lugares. Más información aquí - <https://github.com/casperjs/casperjs/issues/1134>

CAPÍTULO 4

selectores

CasperJS hace un uso intensivo de los selectores con el fin de trabajar con el DOM , Y puede usar de forma transparente ya sea CSS3 o XPath expresiones.

Todos los ejemplos a continuación se basan en el código HTML:

```
<!DOCTYPE html>
<html >
<cabecera >
  <meta charset = "UTF-8" > <título> Mi
  página </título>
</cabecera >
<cuerpo >
  <h1 clase = "Título de la página" > Hola </h1 >
  <ul >
    <li > uno </li >
    <li > dos </li >
    <li > tres </li >
  </ul >
  <pie de página > <pag> © 2012 yo </pag> </pie de página >
</cuerpo >
</html >
```

CSS3

Por defecto, CasperJS acepta cuerdas selectores CSS3 para comprobar elementos dentro del DOM. Para comprobar si el < Clase H1 =

"page-title"> existe el elemento en la página de ejemplo, se puede utilizar:

```
var Casper = exigir( 'Casper' ).crear();

casper.start ( 'Http://domain.tld/page.html' , función () {
  Si ( esta . existe ( 'H1.page-título' )) {
    esta . eco( 'Existe el título' );
```

```
});
```

```
casper.run();
```

O si usted está utilizando el [marco de pruebas](#):

```
casper.test.begin ('Existe el epígrafe' , 1 , función suite (prueba) {
    casper.start ('Http://domain.tld/page.html' , función () {
        (test.assertExists 'H1.page-título' );
    }).correr( función () {
        test.done ();
    });
});
```

Algunos otros métodos de prueba convenientes están confiando en los selectores:

```
casper.test.begin ('La página de pruebas de contenido' , 3 , función suite (prueba) {
    casper.start ('Http://domain.tld/page.html' , función () {
        (test.assertExists 'H1.page-título' ); test.assertSelectorHasText ('H1.page-título' , 'Hola' );
        test.assertVisible ('pie de página');

    }).correr( función () {
        test.done ();
    });
});
```

XPath

Nuevo en la versión 0.6.8. Se puede utilizar como alternativa expresiones XPath en

lugar:

```
casper.start ('Http://domain.tld/page.html' , función () {
    esta . test.assertExists ({
        tipo : 'XPath' , camino : '// * [clase @ = "plop"]'

    }, 'Existe el elemento' );
});
```

Para facilitar el uso y la lectura de las expresiones XPath, una `selectXPath` ayudante está disponible en el Casper módulo:

```
var x = exigir ('Casper') .selectXPath;

casper.start ('Http://domain.tld/page.html' , función () {
    esta . test.assertExists (x ('// * [@ id = "plop"]'), 'Existe el elemento');
});
```

Advertencia: La única limitación del uso de XPath en CasperJS está en el `Casper.fill()` método cuando se desea llenar `fi` campos; PhantomJS nativa sólo permite el uso de selectores CSS3 en su método `uploadfile`, Por lo tanto, esta limitación.

CAPÍTULO 5

Pruebas

CasperJS barcos con su propio [marco de pruebas](#), proporcionar un conjunto puñado de herramientas para facilitar las pruebas de su aplicaciones web.

Advertencia: Cambiado en la versión 1.1.

El marco de pruebas - de ahí su conjunto API - sólo se puede usar cuando se utiliza el prueba casperjs Mand subcom-:

- Si intenta utilizar la casper.test propiedad fuera del entorno de prueba, obtendrá un error;
- A partir de 1.1-beta3, no se puede anular el fi gurada PRECON Casper ejemplo en este entorno de prueba. Puede leer más sobre el por qué de la [Preguntas entrada dedicada](#).

Examen de la unidad

Imagine un tonto Vaca objeto que queremos prueba de unidad:

```
función Vaca() {  
    esta . segado = falso ;  
    esta . mugir = función moo () {  
        esta . segado = cierto ; // Estado mootable: no hagas eso en casa  
        regreso " ¡mugir! " ; } ; }
```

Vamos a escribir un pequeño banco de pruebas para ello:

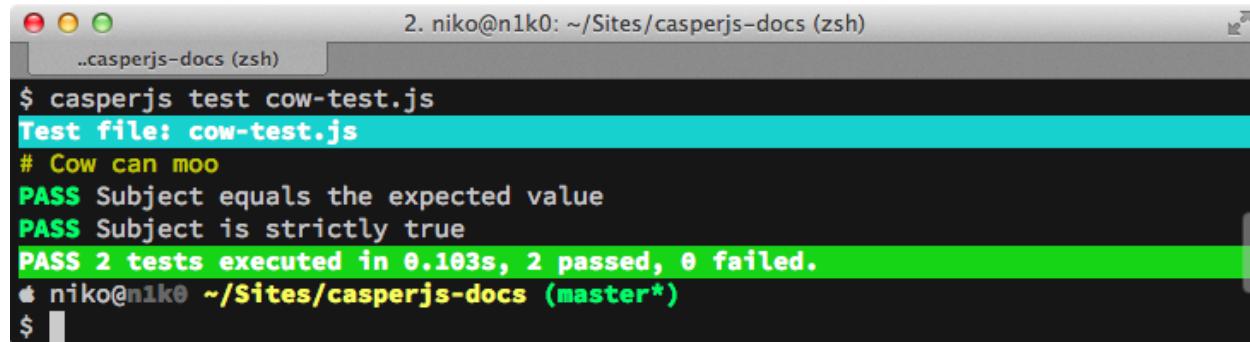
```
// vaca-Test.js  
casper.test.begin ( 'Vaca puede MOO' , 2 , función suite (prueba) {  
    var vaca = nuevo Vaca(); test.assertEquals (cow.moo () , ' ¡mugir! ' );  
    test.assert (cow.mowed);
```

```
test.done ();});
```

Ejecutar las pruebas utilizando el prueba casperjs mando:

```
$ test casperjs vaca-Test.js
```

Usted debe teóricamente obtener algo como esto:



```
2. niko@n1k0: ~/Sites/casperjs-docs (zsh)
..casperjs-docs (zsh)
$ casperjs test cow-test.js
Test file: cow-test.js
# Cow can moo
PASS Subject equals the expected value
PASS Subject is strictly true
PASS 2 tests executed in 0.103s, 2 passed, 0 failed.
$ niko@n1k0 ~/Sites/casperjs-docs (master*)
$
```

Hacer que falle:

```
casper.test.begin ('Vaca puede MOO' , 2 , función suite (prueba) {
    var vaca = nuevo Vaca(); test.assertEquals (cow.moo () , 'Bazinga!' );
    test.assert (cow.mowed); test.done ();});
```

Usted obtendrá este lugar:

Insinuación: el conjunto ensayador API módulo está documentado [aquí](#).

pruebas de navegador

Ahora vamos a escribir una serie de pruebas para la búsqueda de Google (sí, has leído bien):

```
// googletesting.js
casper.test.begin ('Búsqueda de Google recupera 10 o más resultados' , 5 , función
.. suite (prueba) {
    casper.start ( "Http://www.google.fr" , función () {
        test.assertTitle ( "Google" , "Google título página de inicio es el esperado" ); (test.assertExists 'Forma [action = "/ búsqueda"]' , "Forma
principal se encuentra");
        esta . llenar ( "Forma [action = "/ búsqueda]" , {Q : "casperjs"
        }, cierto );
    });

    casper.then ( función () {
        test.assertTitle ( "Casperjs - Recherche Google" , "Google título está bien" ); test.assertUrlMatch ( / Q = casperjs / , "Término de
búsqueda ha sido realizada" );
    });
}
```

```
2. niko@n1k0: ~/Sites/casperjs-docs (zsh)
..casperjs-docs (zsh)

$ casperjs test cow-test.js
Test file: cow-test.js
# Cow can moo
FAIL Subject equals the expected value
#   type: assertEquals
#   file: cow-test.js:11
#   code: test.assertEquals(cow.moo(), 'BAZINGA!');
#   subject: "moo!"
#   expected: "BAZINGA!"
△ Cow can moo: 2 tests planned, 1 tests executed
FAIL 1 tests executed in 0.106s, 0 passed, 1 failed.

Details for the 1 failed test:

In cow-test.js:11
Cow can moo
  assertEquals: Subject equals the expected value
$ 
```

```
test.assertEval ( función () {
    regreso __utils__.findAll ("H3.r").longitud >= 10 ; }, "Google búsqueda de \" casperjs \ "recupera 10 o más resultados");

});

casper.run ( función () {
    test.done (); });
}); 
```

Ahora ejecutar el banco de pruebas:

```
$ googletesting.js prueba casperjs
```

Probablemente obtendrá algo como esto:

```
2. niko@n1k0: ~/Sites/casperjs-docs (zsh)
..casperjs-docs (zsh)

$ casperjs test googletesting.js
Test file: googletesting.js
# Google search retrieves 10 or more results
PASS google homepage title is the one expected
PASS main form is found
PASS google title is ok
PASS search term has been submitted
PASS google search for "casperjs" retrieves 10 or more results
PASS 5 tests executed in 1.249s, 5 passed, 0 failed.
$ 
```

Configuración de las opciones de Casper en el entorno de prueba

Como debe utilizar un `fi` guardada PRECON Casper instancia dentro del entorno de prueba, la actualización de su `opciones` se puede lograr de esta manera:

```
casper.options.optionName = valor de la opción; // donde se desea obviamente el OptionName  
.. nombre de la opción  
  
casper.options.clientScripts.push ( "nuevos"-script.js );
```

L

Técnicas avanzadas

los `Tester #begin()` acepta ya sea una función o un objeto para describir una suite; la opción objeto permite configurar `preparar()` y `demoler()` funciones:

```
// vaca-Test.js  
casper.test.begin ( 'Vaca puede MOO' , 2 , { preparar : función ( prueba ) {  
  
    esta . vaca = nuevo Vaca(); ,  
  
    demoler : función ( prueba ) {  
        esta . cow.destroy (); ,  
  
    prueba : función ( prueba ) {  
        (test.assertEquals esta . cow.moo () , '¡mugir!' ); test.assert ( esta . cow.mowed);  
        test.done (); }});
```

args y opciones de comandos de prueba

argumentos

los prueba casperjs comando tratar a cada argumento pasado como `fi` o rutas de directorios que contienen las pruebas. Se forma recursiva buscará cualquier directorio pasado a buscar `*.js` o `*.café` archivos y añadirlos a la pila.

Advertencia: Hay dos condiciones importantes al escribir pruebas:

- Usted no debe crear un nuevo Casper ejemplo, en una prueba de `fi` le;
- Usted debe llamada `Tester.done()` cuando todas las pruebas contenidas en una suite (o en un `fi` le) han sido ejecutadas.

opciones

Las opciones son pre `fi` jos con un doble guión (-):

- `--xUnit = <nombre de archivo>` exportará resultados de las pruebas en un baño `XUnit XML fi`

- - - directo o - verboso imprimirá *los mensajes de registro* directamente a la consola
- - - log-level = <logLevel> establece el nivel de registro (véase la [sección relacionada](#))
- - - auto-salida = no impide que el corredor de prueba para salir cuando se han ejecutado todas las pruebas; esto normalmente permite realizar operaciones complementarias, aunque implica para salir Casper escuchar manualmente a la salida probador evento:

```
// $ casperjs prueba --auto-salida = sin
casper.test.on( "salida" , función () {
    someTediousAsyncProcess ( función () {
        casper.exit (); });
});
```

Nuevo en la versión 1.0.

- - - incluye = foo.js, bar.js incluirá la foo.js y bar.js archivos antes de cada prueba fi l de ejecución.
- - - pre = pre-Test.js agregará las pruebas contenidas en pre-Test.js antes de la ejecución de toda la serie de pruebas.
- - - post = post-Test.js agregará las pruebas contenidas en post-Test.js después haber ejecutado toda la serie de pruebas.
- - - fail-fast terminará el conjunto de pruebas actual, tan pronto como se detecta un fallo de primera.
- - - conciso creará una salida más concisa de la serie de pruebas.
- - - no-colores creará una salida sin colores (precioso) de casperjs. comando personalizado de

ejemplo:

```
$ Casperjs prueba --includes = foo.js, bar.js \
    - - pre = pre-Test.js \
    - - post = post-Test.js \
    - - \directa \
    - - log-level = debug \
    - - fail-fast \
    test1.js test2.js / ruta / a / some / test / dir
```

Advertencia: En desuso desde la versión 1.1.

- - directo opción se ha cambiado el nombre a - verboso, aunque -- directo se sigue trabajando, mientras que ha de ser considerado obsoleta.

Insinuación: UN lo esencial de demostración También está disponible con el fin de que pueda empezar con un conjunto de muestras que implica algunas de estas opciones.

Exportación de los resultados en formato XUnit

CasperJS pueden exportar los resultados del conjunto de pruebas a un XML XUnit fi l, que es compatible con las herramientas de integración continua como [Jenkins](#). Para guardar el registro XUnit de su suite de prueba, utilice el - xUnit opción:

```
$ Casperjs prueba googletesting.js --xunit = log.xml
```

Usted debe obtener un informe bastante XUnit XML como esto:

```
<? Xml version = "1.0" encoding = "UTF-8"?>
<bancos de pruebas duración = "1,249" >
    <banco de pruebas errores = "0" fallas = "0" name = "Búsqueda Google recupera 10 o más
    .. resultados "paquete = "googletesting" pruebas = "5" tiempo = "1,249" marca de tiempo =" 2012-12-
    .. 30T21: 27: 26.320Z" >
        <caso_prueba nombre de clase = "googletesting" name = "Título de página principal de Google es el
        .. espera" tiempo = '0,813' />
            <caso_prueba classname = "googletesting" name = "forma principal se encuentra" tiempo = "0,002" />
            <caso_prueba nombre de clase = "googletesting" name = "Título de Google está bien" tiempo = "0,416" />
            <caso_prueba nombre de clase = "googletesting" name = "término de búsqueda ha sido realizada"
        .. tiempo = "0,017" />
            <caso_prueba nombre de clase = "googletesting" name = "google búsqueda de & quot; y casperjs
        .. quot; recupera 10 o más resultados" tiempo = '0,001' />
            <Sistema de salida /> </
        banco de pruebas> </ bancos de
        pruebas>
```

Puede personalizar el valor de la *nombre* propiedad haciendo pasar un objeto a `casper.test.fail()` me gusta:

```
casper.test.fail ('Google búsqueda 'casperjs' recupera 10 o más resultados' , {nombre :
    ..'recuento resultado es 10+'});
```

```
<? Xml version = "1.0" encoding = "UTF-8"?>
<bancos de pruebas duración = "1,249" >
    <banco de pruebas errores = "0" fallas = "0" name = "Búsqueda Google recupera 10 o más
    .. resultados "paquete = "googletesting" pruebas = "5" tiempo = "1,249" marca de tiempo =" 2012-12-
    .. 30T21: 27: 26.320Z" >
        <caso_prueba nombre de clase = "googletesting" name = "Título de página principal de Google es el
        .. espera" tiempo = '0,813' />
            <caso_prueba classname = "googletesting" name = "forma principal se encuentra" tiempo = "0,002" />
            <caso_prueba nombre de clase = "googletesting" name = "Título de Google está bien" tiempo = "0,416" />
            <caso_prueba nombre de clase = "googletesting" name = "término de búsqueda ha sido realizada"
        .. tiempo = "0,017" />
            <caso_prueba className = "googletesting" name = "Resultados de recuento es 10+" tiempo = "0,001" />
                <fracaso type = "fracaso" > Google búsqueda "casperjs" recupera 10 o más
        .. resultados </ insuficiencia>
            <Sistema de salida /> </
        banco de pruebas> </ bancos de
        pruebas>
```

CasperJS propias pruebas

CasperJS tiene su propia unidad y conjunto de pruebas funcionales, que se encuentra en el `pruebas` subcarpeta. Para ejecutar este conjunto de pruebas:

```
$ Casperjs autocomprobación
```

Nota: La ejecución de este conjunto de pruebas es una gran manera de encontrar algún error en su plataforma. Si falla, no dude en [fíjese en un problema](#) o preguntar en la [CasperJS lista de correo](#).

La extensión de Casper para Pruebas

Este comando:

```
prueba $ casperjs [ruta]
```

es sólo un atajo para éste:

```
$ Casperjs /path/to/casperjs/tests/run.js [ruta]
```

Así que si desea ampliar las capacidades de Casper para sus pruebas, lo mejor es escribir su propio corredor y extender la instancia del objeto Casper desde allí.

Insinuación: Puede hallar el código predeterminado en corredor [run.js](#).

CAPÍTULO 6

Documentación de la API

Aquí encontramos una referencia bastante completa de la API CasperJS. Si algo es erróneo o faltante, [fíjate](#) de un problema .

los Casper módulo

los Casper clase

La forma más fácil de obtener una instancia Casper es el uso de la década de los módulos crear() método:

```
var Casper = exigir('Casper').crear();
```

Pero también se puede recuperar la función principal y instanciarlo en solitario:

```
var Casper = nuevo exigir('Casper') .Casper();
```

Insinuación: Además, la salida [cómo extender Casper con sus propios métodos](#).

Ambos Casper constructor y el crear() función aceptar un único opciones argumento que es un objeto javascript estándar:

```
var Casper = exigir('Casper') .Create ({verbosa : cierto ,  
nivel de registro : "depurar"  
});
```

Casper.options

Un opciones objeto se puede pasar a la Casper constructor, por ejemplo .:

```
var Casper = exigir('Casper').Create ({clientScripts : [  
    'Incluye / jquery.js' ,  
    'Incluye / underscore.js' // DOM en cada petición  
],  
configuración de página : {  
    cargar imágenes : falso ,  
    loadPlugins : falso  
},  
nivel de registro : "Info" ,  
verboso : cierto  
});
```

También puede modificar las opciones en tiempo de ejecución:

```
var Casper = exigir('Casper').crear(); casper.options.waitForTimeout = 1000 ;
```

La lista completa de las opciones disponibles se detalla a continuación.

clientScripts

Tipo: Formación

Defecto: []

Una colección de lepaths guión fi a incluir en cada página cargada.

exitOnError

Tipo: Boole

Defecto: cierto

Establece si CasperJS deben salir cuando un error no detectada ha sido lanzada por el guión.

httpStatusHandlers

Tipo: Objeto

Defecto: {}

Un objeto JavaScript que contiene funciones que llamar cuando un recurso solicitado tiene un código de estado HTTP dado. Una muestra dedicada se proporciona como un ejemplo.

nivel de registro

Tipo: Cuerda

Defecto: "error"

Nivel de registro (véase la sección de registro para obtener más información)

en alerta

Tipo: Función

Defecto: nulo

Firma: onAlert (Object Casper, mensaje String)

Una función que se llamará cuando se dispara una alerta de JavaScript ()

Ondie

Tipo: Función

Defecto: nulo

Firma: Ondie (Object Casper, String mensaje, el estado de cuerda)

Una función que se llamará cuando Casper # die () se llama

onError

Tipo: Función

Defecto: nulo

Firma: onError (Object Casper, cadena msg, Array traza inversa)

Una función que se llamará cuando se produce un evento de nivel "error"

onLoadError

Tipo: Función

Defecto: nulo

Firma: onLoadError (Objeto Casper, Cadena casper.requestUrl, el estado de cuerda)

Una función que se llamará cuando un recurso solicitado no se puede cargar

onPageInitialized

Tipo: Función

Defecto: nulo

Firma: onPageInitialized (página Object)

Una función que se llamará después Página web instancia se ha inicializado

onResourceReceived

Tipo: Función

Defecto: nulo

Firma: onResourceReceived (Object Casper, recursos Object)

Método de proxy para PhantomJS' Página Web # onResourceReceived () devolución de llamada, pero la instancia actual de Casper se pasa como primer argumento.

onResourceRequested

Tipo: Función

Defecto: nulo

Firma: **onResourceRequested (Object Casper, recursos Object)**

método de proxy para PhantomJS' el Web # onResourceRequested () de devolución de llamada, pero la instancia actual de Casper se pasa como primer argumento.

onStepComplete

Tipo: Función

Defecto: nulo

Firma: **onStepComplete (Object Casper, stepResult)**

Una función que se ejecuta cuando una ejecución de la función paso es terminado.

onStepTimeout

Tipo: Función

Defecto: Función

Firma: **onStepTimeout (entero de tiempo de espera, Integer stepNum)**

Una función que se ejecuta cuando un tiempo de ejecución de la función paso excede el valor de la opción stepTimeout, en su caso se ha establecido.

Por defecto, el tiempo de espera de la secuencia de comandos se cerrará mostrar un error, excepto en el entorno de prueba donde se acaba de añadir un fracaso en los resultados de baño.

onTimeout

Tipo: Función

Defecto: Función

Firma: **onTimeout (entero de tiempo de espera)**

Una función que se ejecutará cuando el tiempo de ejecución del script excede el valor de la opción de tiempo de espera, en caso de haberse establecido. Por defecto, el tiempo de espera de la secuencia de comandos se cerrará mostrar un error, excepto en el entorno de prueba donde se acaba de añadir un fracaso en los resultados de baño.

onWaitTimeout

Tipo: Función

Defecto: Función

Firma: **onWaitTimeout (tiempo de espera entero)**

Una función que se ejecuta cuando una waitFor * tiempo de ejecución de la función excede el valor de la opción WaitTimeout, en caso de haberse establecido.

Por defecto, el tiempo de espera de la secuencia de comandos se cerrará mostrar un error, excepto en el entorno de prueba donde se acaba de añadir un fracaso en los resultados de baño.

página

Tipo: Página web

Defecto: nulo

Un PhantomJS existentes Página web ejemplo

Advertencia: Anulación el página propiedades pueden causar algunas de las características Casper puede no funcionar. Por ejemplo, anulando la onUrlChanged propiedad hará que el waitForUrl característica no funciona.

configuración de página

Tipo: Objeto

Defecto: {}

configuración de la página web de PhantomJS objeto. Los valores disponibles son:

- javascriptEnabled de fi ne si se debe ejecutar la secuencia de comandos en la página o no (por defecto cierto)
- cargar imágenes de fi ne la posibilidad de cargar las imágenes inline o no
- localToRemoteUrlAccessEnabled de multas si los recursos locales (por ejemplo, de fi l) pueden acceder a URLs remotas o no (por defecto falso)

- agente de usuario define el agente de usuario envía al servidor cuando la solicita recursos de páginas web
- nomUsuario establece el nombre de usuario utilizado para la autenticación HTTP
- contraseña configura la contraseña utilizada para la autenticación HTTP

• resourceTimeout (en milisegundos) define el tiempo de espera después del cual cualquier recurso solicitado dejará de intentar y proceder con otras partes de la página.

onResourceTimeout devolución de llamada será llamado en tiempo de espera. unde definido (valor predeterminado) significa parámetros gecko predeterminados. PhantomJS ajustes especí fi cas:

- XSSAuditingEnabled de fi nes si las solicitudes de carga deben ser monitorizados para intentos cross-site de secuencias de comandos (por defecto a falso)
- webSecurityEnabled de fi ne si la seguridad de la tela debe ser habilitado o no (por defecto a true) SlimerJS ajustes especí fi cas:

- allowMedia falsa para desactivar la carga de medios (audio / vídeo). Por defecto: cierto. (Sólo SlimerJS)
- maxAuthAttempts indicar el máximo de intentos de autenticación HTTP. (SlimerJS 0.9)
- plainTextAllContent true para indicar que webpage/plainText devuelve todo, incluso el contenido de los elementos de script, elementos invisibles, etc .. Por defecto: falso.

remoteScripts

Tipo: Formación

Defecto: []

Nuevo en la versión 1.0.

Una colección de secuencias de comandos remotas direcciones URL a incluir en cada página cargada

safeLogs

Tipo: Boole

Defecto: cierto

Nuevo en la versión 1.0.

Cuando esta opción se establece en true -que es el valor por defecto, cualquier información contraseña introducida en <input type ="contraseña"> será ofuscado en los mensajes de registro. Establecer safeLogs a falsa a revelar las contraseñas en texto plano (no se recomienda).

silentErrors

Tipo: Boole

Defecto: falso

Cuando se activa esta opción, errores de etapa atrapados no son lanzados (aunque todavía eventos relacionados son emitidos). Generalmente se utiliza internamente en un contexto de prueba.

stepTimeout

Tipo: Número

Defecto: nulo

paso de tiempo de espera máximo en milisegundos; Cuando se establece, todas las funciones de fi nida paso tendrá que ejecutar antes de que se haya alcanzado este valor de tiempo de espera. Puede de fi nir la devolución de llamada onStepTimeout () para coger un caso así. Por defecto, el guión va a morir () con un mensaje de error.

se acabó el tiempo

Tipo: Número

Defecto: nulo

tiempo de espera máximo en milisegundos

verboso

Tipo: Boole

Defecto: falso

de salida en tiempo real de mensajes de registro

viewportSize

Tipo: Objeto

Defecto: nulo

tamaño de la ventana gráfica, por ejemplo. { anchura: 800, altura: 600}

Nota: PhantomJS barcos con una ventana gráfica de 400x300, y CasperJS no anularán por defecto.

RetryTimeout

Tipo: Número

Defecto: 100

Retraso predeterminado entre los intentos, por Espere* funciones de la familia.

WaitTimeout

Tipo: Número

Defecto: 5000

Predeterminado de tiempo de espera, por Espere* funciones de la familia.

Casper prototipo

espalda()

Firma: espalda()

Se mueve un paso atrás en la historia del navegador:

```
casper.start ( 'Http://foo.bar/1' ) Casper.thenOpen ( 'Http://foo.bar/2' );
casper.thenOpen ( 'Http://foo.bar/3' ); casper.back (); casper.run ( función
() {

    console.log ( esta .getCurrentUrl () ); // ' http://foo.bar/2'
});
```

También eche un vistazo a las [adelante\(\)](#).

base64encode ()

Firma: base64encode (String url [, método String, datos de objeto])

Codifica un recurso mediante el algoritmo de base 64 de forma sincrónica utilizando XMLHttpRequest del lado del cliente.

Nota: No podemos utilizar window.btoa () porque falla estrepitosamente en la versión de la navegación WebKit con Phan- tomJS.

Ejemplo: recuperar imagen logo google codificado en base64:

```
var base64logo = nulo ;
casper.start ('Http://www.google.fr' , función () {
    base64logo = esta . base64encode ('Http://www.google.fr/images/srpr/logo3w.png');
});

casper.run ( función () {
    esta . echo (base64logo) . Salir ();
});
```

También puede realizar una solicitud POST HTTP para recuperar el contenido para codificar:

```
var base64contents = nulo ;
casper.start ('Http://domain.tld/download.html' , función () {
    base64contents = esta . base64encode ('Http://domain.tld/' , 'ENVIAR' , {Param1 : 'Foo' , param2 : 'bar'
}); });

casper.run ( función () {
    esta . echo (base64contents) . Salir ();
});
```

derivación()

Firma: de derivación (nb Numbr)

Nuevo en la versión 1.1.

No pasa por un determinado número de pasos definida de navegación:

```
casper.start (); casper.then ( función () {

    // Este paso se ejecutará
});
casper.then ( función () {
    esta . derivación( 2 );
});
casper.then ( función () {
    // Esta prueba no se ejecutará
});
casper.then ( función () {
    // Tampoco éste
});
casper.run ();
```

hacer clic()

Firma: clic (selector de cadena, [Número | Cadena X, Número | Cadena Y])

Realiza un clic en el elemento que cumpla con la proporcionada *selector de expresión*. El método trata de dos estrategias secuenciales cialmente:

1. tratar de desencadenar una MouseEvent en Javascript

2. usando evento QtWebKit nativo si el anterior intento fracasó

Ejemplo:

```
casper.start ('Http://google.fr');

casper.thenEvaluate ( función ( término) {
    documento .querySelector ( 'De entrada [name = "q"]' ).setAttribute ( 'valor' , término);
    documento .querySelector ( 'Forma [name = "f"]' ).enviar();
}, " CasperJS ");

casper.then ( función () {
    // Haga clic en enlace de primera consecuencia
    esta . hacer clic('H3.ra');
});

casper.then ( función () {
    // Haga clic en enlace de primera consecuencia
    esta . hacer clic('H3.ra' , 10 , 10 );
});

casper.then ( función () {
    // Haga clic en enlace de primera consecuencia
    esta . hacer clic('H3.ra' , "50%" , "50%" );
});

casper.then ( función () {
    console.log ( 'Hecho clic en Aceptar, la nueva ubicación es' + esta .getCurrentUrl ());
};

casper.run ();
```

clickLabel ()

Firma: clickLabel (etiqueta String [, etiqueta String])

Nuevo en la versión 0.6.1.

Hace clic en el elemento DOM primera encontrados que contienen etiqueta texto. Opcionalmente se asegura de que el nombre de nodo elemento es etiqueta:

```
// <a href="#"> Mi vínculo es hermosa </a>
casper.then ( función () {
    esta . clickLabel ( 'Mi vínculo es bella' , 'un' );
});

// <botón type = "submit"> Pero mi botón es más sexy </botón>
casper.then ( función () {
    esta . clickLabel ( 'Pero mi botón es más sexy' , 'botón' );
});
```

capturar()

Firma: capturar (String targetFilepath, [clipRect objeto, imgOptions objeto])

Método de proxy para PhantomJS' Página web # rinde. añade una clipRect parámetro para el ajuste automático de la página
clipRect establecer y vuelve de nuevo una vez hecho:

```
casper.start ('Http://www.google.fr', función () {
    esta . capturar( 'Google.png' , { parte superior : 100 ,
        izquierda : 100 , ancho : 500 , altura : 400

    });
});

casper.run ();
```

Nuevo en la versión 1.1. los imgOptions objeto permite especificar dos

opciones:

- formato para establecer el formato de la imagen manualmente, evitando confiar en el nombre de archivo fi
- calidad para ajustar la calidad de imagen, de 1 a 100 Ejemplo:

```
casper.start ('Http: // foo' , función () {
    esta . capturar( 'Foo' , indefinido , {
        formato : 'Jpg' , la calidad : 75

    });
});
```

captureBase64 ()

Firma: captureBase64 (formato String [, zona mixta])

Nuevo en la versión 0.6.5. Calcula la base64 representación de una captura de imagen binaria de la página actual, o un área dentro de la página, en un formato determinado.

Formatos de imagen soportados son bmp, jpg, jpeg, png, ppm, tiff, xbm y xpm.

los zona argumento puede ser cualquiera de los siguientes tipos:

- Cuerda: zona es un selector CSS3 cuerda, p.ej. div # forma plop [name = "forma"]
de entrada [type = "submit"]
- clipRect: zona es un objeto clipRect, por ejemplo. { " top ": 0, " izquierda ": 0, " ancho ": 320, " altura ": 200}
- Objeto: zona es una *objeto selector*, p.ej. un Ejemplo selector XPath:

```
casper.start ('Http://google.com' , función () {
    // captura selector
    console.log ( esta . captureBase64 ( 'Png' , '#Iga' ));

    // captura clipRect
    console.log ( esta . captureBase64 ( 'Png' , { parte superior : 0 , izquierda : 0 ,
        ancho : 320 , altura : 200

    }));
    // captura de página entera
```

```

    console.log ( esta . captureBase64 ( 'Png' ));
});

casper.run ();

```

captureSelector ()

Firma: captureSelector (String TARGETFILE, el selector String [, de objetos imgOptions])

Captura el área de la página que contiene el selector proporcionado y lo guarda en archivo de destino:

```

casper.start ( 'Http://www.weather.com' , función () {
    esta . captureSelector ( 'Weather.png' , '#Mirando hacia el futuro' );
});

casper.run ();

```

Nuevo en la versión 1.1. los imgOptions objeto permite especificar dos

opciones:

- formato para establecer el formato de la imagen manualmente, evitando depender del objetivo fi chero
- calidad para ajustar la calidad de la imagen, de 1 a 100

claro()

Firma: claro()

Nuevo en la versión 0.6.5.

Borra el contexto actual entorno de ejecución de la página. Útil para evitar tener previamente cargado contenido de DOM ser todavía activo.

Piense en ello como una manera de detener la ejecución javascript dentro del entorno DOM remoto:

```

casper.start ( 'Http://www.google.fr' , función () {
    esta . claro(); // La ejecución de JavaScript en esta página ha sido detenido
});

casper.then ( función () {
    // ...
});

casper.run ();

```

limpiar cache()

Firma: limpiar cache()

Nuevo en la versión 1.1.5.

Reemplazar página actual por un nuevo objeto de página, con nextPage () y borrar la memoria caché, con clearMemoryCache (). Ejemplo:

```
casper.start ('Http://www.google.fr' , función () {
    esta . limpiar cache(); // limpiado la caché de memoria y se sustituye con el objeto de página
    .. nueva pagina();
});

casper.then ( función () {
    //...
});

casper.run ();
```

`clearMemoryCache ()`

Firma: `clearMemoryCache ()`

Nuevo en la versión 1.1.5.

Utilice el motor de page.clearMemoryCache () para borrar la memoria caché. Ejemplo:

```
casper.start ('Http://www.google.fr' , función () {
    esta . clearMemoryCache(); // limpiado la caché de memoria.
});

casper.then ( función () {
    //...
});

casper.run ();
```

`debugHTML ()`

Firma: `debugHTML ([selector String, exterior Boolean])`

Emite los resultados de `getHTML ()` directamente a la consola. Toma los mismos argumentos que `getHTML ()`.

`debugPage ()`

Firma: `debugPage ()`

Registra el contenido textual de la página actual directamente a la salida estándar, para los propósitos de depuración:

```
casper.start ('Http://www.google.fr' , función () {
    esta . debugPage ();
};

casper.run ();
```

`morir()`

Firma: `morir (mensaje String [, int status])`

Sale de fantasma con un mensaje de error registrado y un código de estado de salida opcional:

```
casper.start ('Http://www.google.fr' , función () {
    esta . morir ("Fallar." , 1 );
});

casper.run ();
```

descargar()

Firma: descargar (String url, cadena de destino [, el método de cadena, los datos de objeto])

Guarda un recurso remoto en el sistema de ficheros fi. Opcionalmente, puede establecer el método HTTP utilizando el método argumento, y solicitar pasar argumentos a través de la datos objeto (ver [base64encode \(\)](#)):

```
casper.start ('Http://www.google.fr' , función () {
    var url = 'Http://www.google.fr/intl/fr/about/corporate/company/' ;
    esta . descargar (url, 'Google_company.html' );
});

casper.run ( función () {
    esta . eco ('Hecho.' ).salida();
});
```

Nota: Si usted tiene la descarga de archivos con algunas dificultades, tratar de [seguridad web desactivar](#).

cada()

Firma: cada (array matriz, función fn)

Repite elementos de matriz proporcionadas y ejecutar una devolución de llamada:

```
var campo de golf = [
    'Http://google.com/' ,
    'Http://yahoo.com/' ,
    'Http://bing.com/'
];

casper.start (). cada uno (enlaces, función ( uno mismo, link) {
    self.thenOpen (enlace, función () {
        esta . eco( esta . getTitle ());
    });
});

casper.run ();
```

Insinuación: Echar un vistazo a la [googlematch.js](#) script de ejemplo para un caso concreto.

eachThen ()

Firma: eachThen (array Array, Función continuación)

Nuevo en la versión 1.1.

Repite elementos de matriz proporcionadas y añade un paso a la pila con los datos actuales que se le atribuye:

```
casper.start().eachThen ([ 1 , 2 , 3 ], función ( respuesta ) {  
    esta . echo ( response.data ); }).correr();
```

Aquí hay un ejemplo para la apertura de un conjunto de direcciones URL:

```
var Casper = exigir ( 'Casper' ).crear();  
var URLs = [ 'Http://google.com/' , 'Http://yahoo.com/' ];  
  
casper.start () . eachThen ( URL, función ( respuesta ) {  
    esta . thenOpen ( response.data, función ( respuesta ) {  
        console.log ( 'Abrió' , Response.url );  
    });});  
  
casper.run ();
```

Nota: elemento actual se almacena en el `response.data` propiedad.

eco()

Firma: `echo (mensaje String [, estilo String])`

Imprime algo a la salida estándar, opcionalmente con un poco de color de fantasía (ver el [módulo colorizer](#) para más información):

```
casper.start ( 'Http://www.google.fr' , función () {  
    esta . eco ( 'Título de la página es:' + esta . evaluar( función () {  
        regreso documento .título;  
    }), 'INFO' ); // Serán impresos en verde en la consola  
});  
  
casper.run ();
```

evaluar()

Firma: `evaluar (función fn [, arg1 [, arg2 [...]]])`

Básicamente PhantomJS' página web # evalúan equivalente. Calcula una expresión en el contexto de la página actual DOM:

```
casper.evaluate ( función ( usuario Contraseña ) {  
    documento .querySelector ( '#username' ).valor = nombre de usuario;  
    documento .querySelector ( '#contraseña' ).valor = contraseña;  
    documento .querySelector ( '#enviar' ).hacer clic();  
}, 'Sheldon Cooper' , 'B4z1ng4' );
```

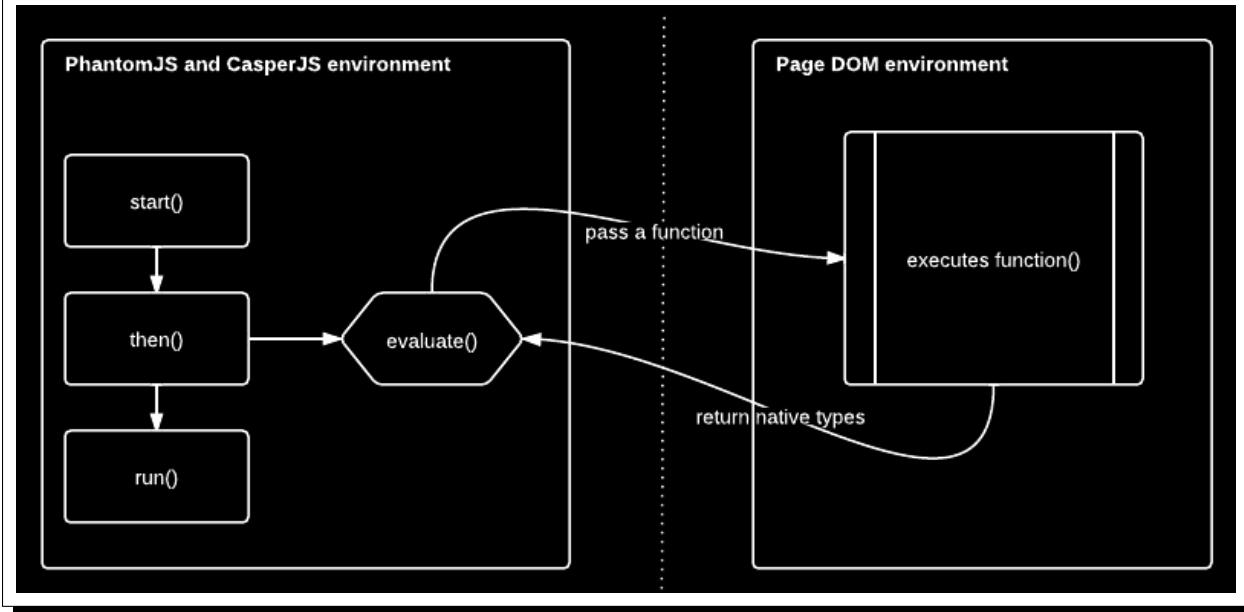
Nota: Para el llenado y presentación de formularios, en lugar utilizar el `fill()` método.

Advertencia: El pre-1.0 manera de pasar argumentos utilizando un objeto se ha mantenido para el propósito de BC, aunque puede no funcionar en algunos casos ; por lo que es recomendable que utilice el método descrito anteriormente.

Comprensión `evaluar()`

El concepto detrás de este método es probablemente el más difícil de entender al descubrir CasperJS. A modo de recordatorio, pensar en el `evaluar()` método como una *portón* entre el medio ambiente CasperJS y el de la página que ha abierto; cada vez que pasa un cierre a `evaluar()`, usted está entrando en la página y ejecutar código como si estuviera usando la consola de navegador.

Aquí está un diagrama elaborado rápidamente tratando de explicar, básicamente, la separación de las preocupaciones:



`evaluateOrDie ()`

Firma: `evaluateOrDie (función fn [, Mensaje de cadena, int status])`

Calcula una expresión dentro de la página actual y DOM morir() si devuelve nada cierto:

```
casper.start ('Http://foo.bar/home', función () {
    esta . evaluateOrDie ( función () {
        regreso / conectado/ .partido( documento .título);
    }, 'no autenticado' );
});

casper.run ();
```

`salida()`

Firma: `de salida ([int status])`

Sale de PhantomJS con un código de estado de salida opcional.

Nota: No se puede confiar en el hecho de que la secuencia de comandos se apagará inmediatamente, ya que este método funciona asíncrona. Esto significa que la secuencia de comandos puede continuar a ser ejecutado después de la llamada de este método. Más información [aquí](#).

existe ()

Firma: existe (selector de cadena)

Comprueba si cualquier elemento dentro de la DOMmatches remotas proporcionado *selector*:

```
casper.start('Http://foo.bar/home', función () {
    Si (esta . existe('#my_super_id')) {
        esta . eco('#my_super_id encontrado', 'INFO');
    } más {
        esta . eco('No #my_super_id encontrado', 'ERROR');
    });
});

casper.run();
```

fetchText ()

Firma: fetchText (selector de cadena)

Recupera contenidos de texto que coinciden con un dado *selector de expresión*. Si usted proporciona un juego más de un elemento, se concatenan sus contenidos textuales:

```
casper.start('Http://google.com/search?q=foo', función () {
    esta . eco(esta . fetchText('H3'));
}).correr();
```

adelante()

Firma: adelante()

Mueve un paso adelante en la historia del navegador:

```
casper.start('Http://foo.bar/1') Casper.thenOpen('Http://foo.bar/2');
casper.thenOpen('Http://foo.bar/3'); casper.back();

// http://foo.bar/2
casper.back(); // http://foo.bar/1
casper.forward(); // http://foo.bar/2
casper.run();
```

También eche un vistazo a las *espalda()*.

Iniciar sesión()

Firma: log (mensaje String [, nivel String, espacio de cadena])

Registra un mensaje con un nivel opcional en un espacio opcional. niveles disponibles son depuración, información, advertencia y error.

Un espacio es una especie de espacio de nombres que puede establecer para filtrado sus registros. Por defecto, Casper registra los mensajes en dos espacios distintos: fantasma y remoto, para distinguir lo que sucede en el entorno PhantomJS desde el mando a distancia uno:

```
casper.start('Http://www.google.fr', función () {
    esta . Iniciar sesión("Estoy de detectar un error", "error");
});

casper.run();
```

llenar()

Firma: llenar (selector de cadena, los valores Object [, Boolean Enviar])

Rellena los campos de un formulario con valores dados y se somete opcionalmente ella. Los campos que son referenciados por su nombre atributo. Cambiado en la versión

1.1: Para utilizar *CSS3 o XPath selectores* En su lugar, comprobar el *llSelectores fi ()* y *llXPath fi ()* métodos. Ejemplo con esta forma html muestra:

```
<formar acción = "/ Contacto" id = "Formulario de contacto" enctype = "/ Form-data multiparte" > <entrada tipo = "Texto" nombre = "tema" /> <textarea  
nombre = "contenido" > </textarea >  
  
<entrada tipo = nombre de "radio" = valor "civilidad" = "Señor" /> Mr <entrada tipo = nombre de "radio" = valor  
"civilidad" = "Señora" /> Sra <entrada tipo = "Texto" nombre = "nombre" /> <entrada tipo = nombre de  
"e-mail" = "correo electrónico" /> <entrada tipo = "nombre del archivo" = "adjunto archivo" /> <entrada tipo = nombre  
de "casilla de verificación" = "Cc" /> Recibir una copia <entrada tipo = "enviar" /> </formar >
```

Una secuencia de comandos para llenar y presentar este formulario:

```
casper.start ('Http://some.tld/contact.form', función () {  
    esta . llenar( 'Forma # formulario de contacto' , {  
        'tema' : 'Te estoy vigilando' ,  
        'contenido' : 'Así que ten cuidado.' ,  
        'civilidad' : 'Señor' ,  
        'nombre' : 'Chuck Norris' ,  
        'correo electrónico' : 'Chuck@norris.com' ,  
        'Cc' : cierto ,  
        'adjunto archivo' : '/Users/chuck/roundhousekick.doc'  
    }, cierto );  
});  
  
casper.then ( función () {  
    esta . evaluateOrDie ( función () {  
        regreso / mensaje enviado/ .prueba( documento .body.innerText);  
    }, 'Envío de mensajes falló' );  
});  
  
casper.run ( función () {  
    esta . eco('mensaje enviado').salida();  
});
```

los llenar() método admite selección individuales en la misma forma que la entrada de texto. Para múltiples selecciona, suministrar una matriz de valores para que coincida contra:

```
<formar acción = "/ Contacto" id = "Formulario de contacto" enctype = "/ Form-data multiparte" > <seleccionar nombre múltiple = "categoría" >  
    <opción valor = "0" Amigos> </opción >  
  
    <opción valor = "1" > Familia </opción >  
    <opción valor = "2" > Acquaintances </opción >  
    <opción valor = "3" > Los colegas </opción >  
    </seleccionar >  
</formar >
```

Una secuencia de comandos para seleccionar varias opciones para la categoría en esta forma:

```
casper.then ( función () {
    esta . llenar( 'Forma # formulario de contacto' , {
        'categorías' : [ '0' , '1' ] // Amigos y familia
   }); });
});
```

Advertencia:

1. El `llenar()` método actualmente no puede llenar si los campos utilizando selectores XPath; PhantomJS forma nativa sólo permite el uso de selectores CSS3 en su subir archivo() método, por lo tanto, esta limitación.
2. Por favor No utilice ni CasperJS PhantomJS para enviar correo no deseado, o voy a estar llamando a la tirada. Más en serio, por favor, simplemente no lo hacen.

fillSelectors ()

Firma: `fillSelectors (selector de cadena, los valores de objeto [, Boolean Enviar])`

Nuevo en la versión 1.1.

Rellena los campos de formulario con valores dados y se somete opcionalmente ella. Los campos que son referenciados por CSS3 selectores:

```
casper.start ( 'Http://some.tld/contact.form' , función () {
    esta . fillSelectors ( 'Forma # formulario de contacto' , {
        'De entrada [name = "sujeto"]' : 'Te estoy vigilando' ,
        'De entrada [name = "contenido"]' : 'Así que ten cuidado.' ,
        'De entrada [name = "cortesía"]' : 'Señor' ,
        'De entrada [name = "name"]' : 'Chuck Norris' ,
        'Entrada [name = "email"]' : 'Chuck@norris.com' ,
        'De entrada [name = "cc"]' : cierto ,
        'De entrada [name = "unión"]' : '/Users/chuck/roundhousekick.doc'
    }, cierto );
});
```

fillLabels ()

Firma: `fillLabels (selector de cadena, los valores Object [, Boolean Enviar])`

Nuevo en la versión 1.1.

Rellena un formulario con valores de campo proporcionado si utilizando texto de la etiqueta asociada campos que son referenciados por los valores de contenido de la etiqueta:

```
casper.start ( 'Http://some.tld/contact.form' , función () {
    esta . fillLabels ( 'Forma # formulario de contacto' , { Email :
        'Chuck@norris.com' ,
        Contraseña : 'arrojar' ,
        Contenido : 'Estoy viendo tú' ,
        Comprobar : cierto ,
        No : cierto ,
        Tema : 'bar' ,
        sobre múltiples temas : [ 'bar' , 'coche' ],
        Archivo : fpath,
});
```

```

    "1" : cierto ,
    "3" : cierto ,
    Extraño : "muy"
}, cierto );
});

```

fillXPath ()

Firma: (Selector de cadena, los valores Object [, Boolean Enviar]) fillXPath

Nuevo en la versión 1.1.

Rellena los campos de formulario con valores dados y se somete opcionalmente ella. Mientras que la formar elemento siempre es referenciado por un selector CSS3, los campos son referenciados por XPath selectores:

```

casper.start('Http://some.tld/contact.form', función () {
  esta.fillXPath('Forma # formulario de contacto', {
    '// input [@ name = "sujeto"] : 'Te estoy vigilando' ,
    '// input [@ name = "contenido"] : 'Así que ten cuidado.' ,
    '// input [@ name = " cortesía "] : 'Señor' ,
    '// input [@ name = " name "] : 'Chuck Norris' ,
    '// entrada [@ name = " email "] : 'Chuck@norris.com' ,
    '// input [@ name = " cc "] : cierto ,
  }, cierto );
});

```

Advertencia: los fillXPath () método actualmente no puede llenar fi l fi campos utilizando selectores XPath; PhantomJS nativa sólo permite el uso de selectores CSS3 en su subir archivo() método, por lo tanto, esta limitación.

getCurrentUrl ()

Firma: getCurrentUrl ()

Recupera URL de la página actual. Tenga en cuenta que la dirección URL será decodificado url:-

```

casper.start('Http://www.google.fr', función () {
  esta.eco(esta.getCurrentUrl()); // "Http://www.google.fr"
});

casper.run();

```

getElementAttribute ()

Firma: getElementAttribute (selector de cadena, atributo de cadena)

Nuevo en la versión 1.0.

Recupera el valor de un atributo en el primer elemento que coincide con el proporcionado *selector* :

```

var Casper = exigir('Casper').crear();

casper.start('Http://www.google.fr', función () {
  exigir('utils').tugurio(esta.getElementAttribute('Div [title = "Google"]', 'título'));/
  "google"
});

```

```
});  
  
casper.run();
```

getElementsAttribute ()

Firma: getElementsAttribute (selector de cadena, atributo de cadena)

Nuevo en la versión 1.1.

Recupera los valores de un atributo en cada elemento que coincide con el proporcionado *selector*:

```
var Casper = exigir('Casper').crear();  
  
casper.start('Http://www.google.fr', función () {  
    exigir('utils').tugurio(esta . getElementsAttribute('Div [title = "Google"]', 'título'));  
    // Google J  
});  
  
casper.run();
```

getElementBounds ()

Firma: getElementBounds (selector de cadena, la página de Boole)

Recupera los límites de un elemento DOM que coincide con el previsto *selector*. Si usted tiene cuadros o / y marcos flotantes, ajuste 'verdadero' para el parámetro de página.

Devuelve un objeto con cuatro teclas: superior, izquierda, ancho y altura, o nulo si el selector no existe:

```
var Casper = exigir('Casper').crear();  
  
casper.start('Http://www.google.fr', función () {  
    exigir('utils').tugurio(esta . (getElementBounds 'Div [title = "Google"]'));  
});  
  
casper.run();
```

Esto sería algo así como:

```
{  
    "altura": 95,  
    "izquierda": 352,  
    "parte superior": diecisés,  
    "anchura": 275  
}
```

getElementsBounds ()

Firma: getElementsBounds (selector de cadena)

Nuevo en la versión 1.0.

Recupera una lista de límites para todos los elementos DOM que concuerden con la proporcionada *selector*.

Devuelve una matriz de objetos con cuatro teclas: superior, izquierda, ancho y altura (ver *getElementBounds ()*).

getElementInfo ()

Firma: getElementInfo (selector de cadena)

Nuevo en la versión 1.0.

Recupera información sobre el primer elemento que coincide con el proporcionado *selector*:

```
casper.start('Http://google.fr', función () {
    exigir('utils').tugurio(esta . getElementInfo ('#hlogo'));
});
```

Da algo así como:

```
{
    "atributos": {
        "alinear": "izquierda",
        "Dir": "Ltr",
        "carné de identidad": "Hlogo",
        "Onload": "Window.lo && lo ()",
        "estilo": "Altura: 110px; anchura: 276px; fondo: url (/images/srpr/logo1w.png) NO-
        .. repetir",
        "título": "Google"
    },
    "altura": 110,
    "Html": "<div nowrap \\> nowrap \\> style = \\> color: # 777; font-size: 16px; Font-
    .. weight: bold; position: relative; left: 214px; superior: 70px \\> Francia </ div>",
    "nombre del nodo": "Div",
    "etiqueta": "<Div dir = \"LTR \\> title = \\> Google \\> align = \\> izquierda \\> id = \\> hlogo \\> onload = \\>
    .. window.lo & amp; & amp; lo () \\> style = \\> altura: 110px; ancho: 276px; background: url (/
    .. images / SRPR / logo1w.png) no-repeat \\> <div nowrap \\> nowrap \\> style = \\> color: # 777; Font-
    .. Tamaño: 16px; font-weight: bold; position: relative; izquierda: 214px; top: 70px \\> Francia </ div> <
    .. div>",
    "texto": "Francia \\n",
    "visible": cierto,
    "anchura": 276,
    "x": 62,
    "Y": 76
}
```

Nota: Este método no devolver un elemento DOM, solamente un simple representación del objeto de la misma; Esto es porque el entorno de Casper no tiene acceso directo a la página de raspado uno.

getElementsInfo ()

Firma: getElementsInfo (selector de cadena)

Nuevo en la versión 1.1.

Recupera información acerca de todos los elementos que coinciden con el proporcionado *selector*:

```
casper.start('Http://google.fr', función () {
    exigir('utils').tugurio(esta . getElementsInfo ('#hlogo'));
});
```

Da algo así como:

```
[  
  {  
    "atributos": {  
      "alinear": "izquierda",  
      "Dir": "Ltr",  
      "carácter de identidad": "HLogo",  
      "Onload": "Window.lol && lol()",  
      "estilo": "Altura: 110px; ancho: 276px; background: url (/images/SRPR/logo1w.  
      ..png) no-repeat",  
      "título": "Google"  
    },  
    "altura": 110,  
    "Html": "<Div nowrap = \"nowrap\" style = \" color: # 777; font-size: 16px; Font-  
    .. weight: bold; position: relative; left: 214px; superior: 70px \"> Francia </ div>",  
    "nombre del nodo": "Div",  
    "etiqueta": "<Div dir = \"LTR\" title = \"Google\" align = \"izquierda\" id = \"hLogo\"  
    .. onload = \"window.lol & amp; & amp; lol ()\" style = \"altura: 110px; ancho: 276px;  
    .. background: url (/images/srpr/logo1w.png) no-repeat \"> <div nowrap = \"estilo nowrap\"=\\  
    .. color: # 777; font-size: 16px; font-weight: bold; position: relative; left: 214px; superior: 70px \">  
    .. Francia </ div> </ div>",  
    "texto": "Francia \\ n",  
    "visible": cierto,  
    "anchura": 276,  
    "x": 62,  
    "Y": 76  
  }]  
]
```

Nota: Este método no devolver una NodeList, solamente un simple conjunto de representaciones de objetos de elementos a juego; Esto es porque el entorno de Casper no tiene acceso directo a la página de raspado uno.

getFormValues ()

Firma: getFormValues (selector de cadena)

Nuevo en la versión 1.0.

Recupera una forma dada todos sus valores de campo:

```
casper.start ('Http://www.google.fr', función () {  
  esta . llenar ('formar', {Q: 'play'}, falso);  
  esta . eco (esta . (getFormValues 'formar') .q); // 'play'  
});  
  
casper.run();
```

getGlobal ()

Firma: getGlobal (String name)

Recupera un valor de la variable global dentro del entorno de DOM remoto por su nombre. Básicamente, getGlobal ('foo') recuperará el valor de window.foo desde la página:

```
casper.start('Http://www.google.fr', función () {
    esta . eco(esta . getGlobal('InnerWidth')); // 1024
});

casper.run();
```

getHTML ()

Firma: getHTML ([selector String, exterior Boolean])

Nuevo en la versión 1.0.

Recupera el código HTML de la página actual. Por defecto, se da salida a todo el contenido de la página HTML:

```
casper.start('Http://www.google.fr', función () {
    esta . eco(esta . getHTML ());
});

casper.run();
```

los getHTML () método también puede volcar el contenido HTML que coincide con una determinada `selector`; por ejemplo, con este código HTML:

```
< html >
  < cuerpo >
    < h1 carné de identidad = "Foobar" > Plop </ h1 >
  </ cuerpo >
</ html >
```

Se puede recuperar esos contenidos utilizando:

```
casper.start('Http://www.site.tld', función () {
    esta . eco(esta . getHTML('H1 # foobar')); // => ' Plop'
});
```

los exterior argumento permite recuperar el contenido HTML exteriores del elemento coincidente:

```
casper.start('Http://www.site.tld', función () {
    esta . eco(esta . getHTML('H1 # foobar', cierto)); // => '< h1 id = "foobar" > Plop </ h1 >'
```

getPageContent ()

Firma: getPageContent ()

Nuevo en la versión 1.0.

Recupera el contenido de la página actual, que trata de otros tipos de contenido que no sea HTML exóticos:

```
var Casper = exigir('Casper').crear();

casper.start().entonces( función () {
    esta . abierto('Http://search.twitter.com/search.json?q=casperjs', {Método : 'obtener', encabezados : {

        'Aceptar' : 'Application / json'
```

```
});  
});  
  
casper.run ( función () {  
    exigir('utils') .dump (JSON.parse ( esta .getPageContent ()));  
    esta . salida(); });
```

getTitle ()

Firma: getTitle ()

Recupera título de la página actual:

```
casper.start ('Http://www.google.fr' , función () {  
    esta . eco( esta . getTitle () ); // " google"  
});  
  
casper.run ();
```

mouseEvent ()

Firma: mouseEvent (tipo String, selector de cadena, [Número | Cadena X,
Número | Cadena Y])

Nuevo en la versión 0.6.9.

Desencadena un evento del ratón sobre el primer elemento que se ajusten el selector proporcionado. eventos son compatibles mouseup, mousedown, clic, dblclick, mousemove, al pasar el ratón, mouseout y para PhantomJS > = 1.9.8 MouseEnter, mouseleave y Menú de contexto.

Advertencia: La lista de eventos soportados depende de la versión del motor en uso. Los motores más antiguos sólo proporcionan apoyo parcial. Para un mejor apoyo el uso reciente obra de PhantomJS o SlimerJS ::

```
casper.start ('Http://www.google.fr' , función () {  
    esta . mouseEvent ('hacer clic' , 'H2 un' , "20%" , "50%");  
});  
  
casper.run ();
```

nueva pagina()

Firma: nueva pagina()

Nuevo en la versión 1.1.

Sólo está disponible desde la versión 1.1.0.

Crea una nueva instancia de página Web:

```

casper.start ('Http://google.com', función () {
    // ...
});

casper.then ( función () {
    casper.page = casper.newPage (); casper.open ('Http://yahoo.com').entonces( función () {

        // ....
    });
});

casper.run ();

```

`abierto()`

Firma: abierta (String ubicación, en Configuración del objeto)

Realiza una petición HTTP para la apertura de un lugar determinado. Puede forjar GET, POST, PUT, DELETE y CABEZA peticiones. Ejemplo para una norma OBTENER solicitud:

```

casper.start ();

casper.open ('Http://www.google.com/').entonces( función () {
    esta . eco('Lo tengo.');
});

casper.run ();

```

Ejemplo para una ENVIAR solicitud:

```

casper.start ();

casper.open ('Http://some.testserver.com/post.php', {Método : 'enviar', los datos :

    {
        'título' : 'Plaf',
        'cuerpo' : 'Guau.'
    });
};

casper.then ( función () {
    esta . eco('Publicado.');
});

casper.run ();

```

Para pasar parámetros anidados arrays:

```

casper.open ('Http://some.testserver.com/post.php', {Método : 'enviar', los datos : {

    'Standard_param' : 'Foo',
    'Nested_param []' : [ // note el uso de corchetes!
        'Alguna cosa',
        'Algo más'
    ]
}}

```

```
});
```

Nuevo en la versión 1.0.

Para enviar algunos datos con codificación UTF-8:

```
casper.open ('Http://some.testserver.com/post.php' , {Método : 'enviar' , encabezados : {  
  
    'Tipo de contenido' : 'Application / json; charset = utf-8'  
},  
codificación : 'UTF-8' , // no se aplican de manera predeterminada  
datos : {  
    'Table_flip' : '("o"  
});
```

Nuevo en la versión 1.1.

También puede establecer cabeceras de petición personalizados para enviar al realizar una petición de salida, pasando la encabezados opción:

```
casper.open ('Http://some.testserver.com/post.php' , {Método : 'enviar' , los datos :  
  
    {  
        'título' : 'Plaf' ,  
        'cuerpo' : 'Guau.'  
    },  
    encabezados : {  
        'Accept-Language' : 'Fr, fr-fr; q = 0.8, en-us; q = 0.5, en; q = 0,3'  
    });
```

recargar()

Firma: recarga ([Función entonces])

Nuevo en la versión 1.0.

Vuelve a cargar la página actual ubicación:

```
casper.start ('Http://google.com' , función () {  
    esta . eco( "cargado" );  
    esta . recargar( función () {  
        esta . eco( "Cargado de nuevo" );  
    });});  
  
casper.run();
```

repetir()

Firma: repetición (int veces, la función de entonces)

Repite una navegación paso a un número determinado de veces:

```
casper.start().repetición(3, función() {
    esta . eco("Tejón");
});

casper.run();
```

resourceExists ()

Firma: resourceExists (String | Función | prueba RegExp)

Comprueba si un recurso ha sido cargado. Puede pasar una función, una cadena o una RegExp ejemplo para realizar la prueba:

```
casper.start('Http://www.google.com/', función() {
    Si (esta . (resourceExists 'Logo3w.png')) {
        esta . eco('Google logo de carga');
    } más {
        esta . eco('Logotipo de Google no se ha cargado', 'ERROR');
    });
};

casper.run();
```

Nota: Si desea esperar a que un recurso a ser cargado, utilice el *waitForResource ()* método.

correr()

Firma: run (fn onComplete [, tiempo int])

Corre todo el conjunto de medidas y, opcionalmente, ejecuta una devolución de llamada cuando todos ellos han sido hechas. Obviamente, llamar a este método es obligatorio con el fin de ejecutar el banco de navegación Casper. suite de Casper no se ejecutará:

```
casper.start('Http://foo.bar/home', función() {
    // ...
});

// bueno, le falta .run () aquí!
```

suite de Casper correrá:

```
casper.start('Http://foo.bar/home', función() {
    // ...
});

casper.run();
```

Casper.run () También acepta una onComplete devolución de llamada, que se puede considerar como un paso final personalizado para realizar cuando se han ejecutado todos los otros pasos. Eso sí, no olvide salida() Casper si definimos un !:

```
casper.start('Http://foo.bar/home', función() {
    // ...
});
```

```
casper.then ( función () {
    // ...
});

casper.run ( función () {
    esta . eco( "Así que toda la suite terminó. );
    esta . salida(); // <-- no me olvides!
});
```

La unión a una devolución de llamada complete.error activará cuando el onComplete devolución de llamada falla.

scrollTo ()

Firma: scrollTo (Número de x, Número y)

Nuevo en la versión 1.1-beta3.

Documento se desplaza actual en las coordenadas definido por el valor de x y y:

```
casper.start ( 'Http://foo.bar/home' , función () {
    esta . scrollTo ( 500 , 300 );
});
```

Nota: Esta operación es sincrónico.

scrollToBottom ()

Firma: scrollToBottom ()

Nuevo en la versión 1.1-beta3.

Desplaza documento actual en su parte inferior:

```
casper.start ( 'Http://foo.bar/home' , función () {
    esta . scrollToBottom ();
});
```

Nota: Esta operación es sincrónico.

Sendkeys ()

Firma: Sendkeys (selector Selector, claves de cadena [,] Opciones de objeto)

Nuevo en la versión 1.0.

Envía eventos de teclado nativos del elemento que cumpla con la proporcionada `selector`:

```
casper.then ( función () {
    esta . Sendkeys ( 'De entrada form.contact # nombre' , 'Duque' );
    esta . Sendkeys ( 'Área de texto form.contact # mensaje' , "Maldita sea, estoy en buen estado." );
```

```
esta . hacer clic('De entrada form.contact [type = "submit"]');
});
```

Nuevo en la versión 1.1.

Los HTMLElements actualmente soportados que pueden recibir eventos de teclado desde Sendkeys son <input>, <textarea>, y cualquier HTMLElement con el atributo contenteditable = "true".

opciones

- (Booleano) de reinicio:

Nuevo en la versión 1.1-beta3. Cuando se establece en cierto, esta opción primer vacía el valor de campo actual. De manera predeterminada, se establece en falso y

Sendkeys () se acaba de añadir cadena en el valor de campo actual.

- (Booleano) keepFocus: Sendkeys () de forma predeterminada se eliminará el enfoque en la entrada de texto campos, que normalmente cerca de autocompletar widgets. Si desea mantener el enfoque, utilice el keepFocus opción. Por ejemplo, si se utiliza jQuery-UI, puede hacer clic en la primera sugerencia de autocompletar usando:

```
casper.then ( función () {
    esta . Sendkeys ('De entrada form.contact # nombre' , 'acción' , {KeepFocus : cierto });
    esta . hacer clic('Form.contact ul.ui-autocompletar li.ui-elemento de menú: de primer hijo un' );
});
```

- (String) modificadores: Sendkeys () acepta una modificadores opción para apoyar modificadores fi clave. Las opciones es una cadena que representa la composición de modificadores Fi para usar, separados por el carácter +:

```
casper.then ( función () {
    esta . Sendkeys ('documento' , 'S' , {modificadores : 'Ctrl + alt + shift' });});
```

Los modificadores disponibles fi son:

- ctrl
- alt
- cambio
- meta
- teclado

setHttpAuth ()

Firma: setHttpAuth (String nombre de usuario, contraseña String)

conjuntos HTTP_AUTH_USER y HTTP_AUTH_PW Los valores para los sistemas de autenticación basados en HTTP:

```
casper.start ();
casper.setHttpAuth ('Sheldon Cooper' , 'B4z1ng4');
```

```
casper.thenOpen ('Http://password-protected.domain.tld' , función () {
    esta . eco( "Estoy en. Bazinga" . );
})
casper.run ();
```

Por supuesto se puede pasar directamente a la cadena de autenticación en el URL para abrir:

```
var url = 'Http://sheldon.cooper: b4z1ng4@password-protected.domain.tld' ;

casper.start (url, función () {
    esta . eco( "Estoy en. Bazinga" . );
})
casper.run ();
```

setMaxListeners ()

Firma: setMaxListeners (entero maxListeners)

Establece el número máximo de oyentes que se pueden agregar para cada tipo de oyente:

```
(casper.setMaxListeners 12 );
```

Nota: registro incorrecto de los oyentes en sus scripts Casper puede resultar en un mensaje de advertencia que indica que se ha detectado una posible fuga EventEmitter. Asegúrese de que está añadiendo oyentes en la forma requerida.

Si necesita un oyente que será procesado por todos los scripts a continuación, asegurar que sólo se ha registrado una sola vez. Si es necesario agregar un oyente por suite de prueba, por ejemplo, asegurar que se añadió al oyente durante la configuración y se retira durante tearDown. Ver

Tester # begin () Para la configuración y la estructura tearDown.

Advertencia: No se recomienda cambiar los oyentes máximo. Sólo se debe aumentar si se necesitan más de la cantidad predeterminada de oyentes en su caso. El aumento de este límite aumentará para todos los tipos de escucha y podría dar lugar a posibles fugas EventEmitter no sea detectada. Si tiene que aumentar este límite, se aumentará en pequeños incrementos.

comienzo()

Firma: iniciar (String url [, a continuación, Función])

cifras estafadores y comienza Casper, a continuación, abre la proporcionada url y añade opcionalmente la etapa proporcionado por el entonces argumento:

```
casper.start ( 'Http://google.fr' , función () {
    esta . eco( "Estoy cargado." );
});
casper.run ();
```

Alternativamente:

```
casper.start ('Http://google.fr');

casper.then ( función () {
    esta . eco( "Estoy cargado." );
});

casper.run ();
```

O alternativamente:

```
casper.start ('Http://google.fr');

casper.then ( función () {
    casper.echo ( "Estoy cargado." );
});

casper.run ();
```

¡Cuestión de gusto!

Nota: Debe llamar al comienzo() método con el fin de poder agregar los pasos de navegación y ejecutar la suite. Si no lo hace obtendrá un mensaje de error que le invita a hacerlo de todos modos.

estado()

Firma: estado (BooleanAsString)

Nuevo en la versión 1.0.

Devuelve el estado de la instancia actual de Casper:

```
casper.start ('Http://google.fr' , función () {
    esta . eco( esta . estado( cierto ) );
});

casper.run ();
```

switchToFrame ()

Firma: switchToFrame (String | Número frameInfo)

Nuevo en la versión 1.1.5.

Cambia la página principal de la trama que tiene el nombre o el número de trama de índice coincidente el argumento pasado. Inyectar secuencias de comandos locales, secuencias de comandos remotos y utilidades de cliente en este marco.

switchToMainFrame ()

Firma: switchToMainFrame ()

Nuevo en la versión 1.1.5.

Cambiar la página principal para el marco principal de la que actualmente está activo.

switchToParentFrame ()

Firma: switchToParentFrame ()

Nuevo en la versión 1.1.5.

Para cambiar la página principal en el bastidor principal.

entonces()

Firma: entonces (Función continuación)

Este método es la forma estándar de añadir un nuevo paso de navegación a la pila, proporcionando una función simple:

```
casper.start ('Http://google.fr');

casper.then ( función () {
    esta . eco( "Estoy en tu Google." );
});

casper.then ( función () {
    esta . eco('Ahora, vamos a escribir algo' );
});

casper.then ( función () {
    esta . eco('Oh bien.' );
});

casper.run ();
```

Puede añadir tantos pasos como sea necesario. Tenga en cuenta que la corriente Casper instancia se une automáticamente el `esta` palabra clave para usted dentro de las funciones escalonadas. Para ejecutar todos los pasos que define, llame al `correr()` método, y listo.

Nota: Debes `comienzo()` la instancia casper el fin de utilizar la `entonces()` método.

Acceso a la respuesta HTTP actual

Nuevo en la versión 1.0.

Se puede acceder al objeto respuesta HTTP actual utilizando el parámetro primero de su devolución de llamada paso:

```
casper.start ('Http://www.google.fr' , función ( respuesta ) {
    exigir( 'utils' ) .dump ( respuesta );
});
```

Eso da:

```
$ casperjs vertedera-headers.js {

  "ContentType": "text / html; charset = UTF-8", "encabezados": [
    {
      "nombre fecha",
      "Valor": "Jue 18 Oct 2012 08:17:29 GMT"}, {

        "Name": "Expires", "valor": "-1"},

      // ... un montón de otras cabeceras],

      "Id": 1,
      "RedirectUrl": null, "etapa": "extremo",
      "status": 200, "statusText": "OK"

      "Tiempo": "2012-10-18T08: 17: 37.068Z", "url":
      "http://www.google.fr/"}
```

Así que a buscar una cabecera en particular por su nombre:

```
casper.start ('Http://www.google.fr' , función ( respuesta ) {
  esta . echo (response.headers.get ('Fecha'));
});
```

Eso da:

```
$ casperjs vertedera-headers.js Jue 18 Oct 2012
08:26:34 GMT
```

Advertencia: funciones paso añaden a *entonces()* se procesan en dos casos diferentes:

1. cuando la función paso anterior ha sido ejecutado,
2. Cuando la solicitud HTTP principal anterior se ha ejecutado y la página *cargado*;

Notar que no hay una única definición de *página cargada*; es que cuando el *domready* evento se ha disparado? Es que "todas las peticiones de ser terminado"? Lo es * toda la lógica de aplicación que se lleva a cabo"? O "todos los elementos que se quedan"? La respuesta siempre depende del contexto. De ahí por qué es recomendable que utilice siempre el *waitFor()* métodos familiares para mantener el control explícito sobre lo que realmente esperas. Un truco común es utilizar *waitForSelector()*:

```
casper.start ('Http://my.website.com');

casper.waitForSelector ('#plop' , función () {
  esta . eco( "Estoy seguro de #plop está disponible en el DOM" );
});

casper.run();
```

thenBypass ()

Firma: thenBypass (Número nb)

Nuevo en la versión 1.1.

Agrega un paso de navegación que pasar por alto un número determinado de pasos siguientes:

```
casper.start ('Http://foo.bar'); casper.thenBypass ( 2 );
casper.then ( función () {

    // Esta prueba no se ejecutará
});
casper.then ( función () {
    // Tampoco éste
});
casper.then ( función () {
    // Si bien esto lo hará
});
casper.run();
```

thenBypassIf ()

Firma: thenBypassIf (condición mixta, Número nb)

Nuevo en la versión 1.1.

Omitir un número dado de pasos de navegación si la condición proporcionada es Truthy o es una función que devuelve un valor Truthy:

```
var universo = {
    responder : 42
};
casper.start ('Http://foo.bar'); casper.thenBypassIf ( función ()
{
    regreso universo && universe.answer === 42 ; }, 2 );

casper.then ( función () {
    // Este paso no se ejecutará como universe.answer es 42
});
casper.then ( función () {
    // Tampoco éste
});
casper.then ( función () {
    // Si bien esto lo hará
});
casper.run();
```

thenBypassUnless ()

Firma: thenBypassUnless (condición mixta, Número nb)

Nuevo en la versión 1.1. Opuesto de [thenBypassIf](#)

).

thenClick ()

Firma: thenClick (selector String [, a continuación, Función])

Añade un nuevo paso de navegación para hacer clic en un selector dado y opcionalmente añadir un nuevo paso de navegación en una sola operación:

```
// Haga clic en el primer eslabón de la página casperJS
casper.start ('Http://casperjs.org') .thenClick ('un', función () {
    esta . eco( "He hecho clic en el primer enlace que se encuentra, la página se ha cargado." );
});

casper.run();
```

Este método es básicamente un atajo conveniente para encadenar un [entonces\(\)](#) y un [hacer clic\(\)](#) llamadas.

thenEvaluate ()

Firma: thenEvaluate (función fn [, arg1 [, arg2 [...]]])

Añade un nuevo paso de navegación para realizar la evaluación código dentro de la corriente DOM página recuperada:

```
// Consultando "Chuck Norris" en Google
casper.start ('Http://google.fr') .thenEvaluate ( función ( término ) {
    documento .querySelector ('De entrada [name = "q"]') .setAttribute ('valor', término);
    documento .querySelector ('Forma [name = "t"]') .enviar();
}, 'Chuck Norris');

casper.run();
```

Este método es un atajo conveniente para encadenar [entonces\(\)](#) y [evaluar\(\)](#) llamadas.

thenOpen ()

Firma: thenOpen (cadena de ubicación [, opciones mixtas])

Añade un nuevo paso de navegación para la apertura de una nueva ubicación, y opcionalmente añadir un siguiente paso cuando su cargado:

```
casper.start ('Http://google.fr') .entonces( función () {
    esta . eco( "Estoy en tu Google." );
});

casper.thenOpen ('Http://yahoo.fr', función () {
    esta . eco( "Ahora estoy en el yahoo." )
});

casper.run();
```

Nuevo en la versión 1.0.

También puede especificar configuración de la solicitud pasando un objeto de configuración (vea [abierta\(\)](#)) como segundo argumento:

```
casper.start (). thenOpen ('Http://url.to/some/uri', {Método : "enviar", los datos : {

    nombre de usuario : 'arrojar',
    contraseña : 'N0rr15'
}})
```

```
}, función () {
    esta . eco( "Solicitud POST se ha enviado." )
});

casper.run();
```

thenOpenAndEvaluate ()

Firma: thenOpenAndEvaluate (ubicación String [, a continuación, la función [, arg1 [, arg2 [, . . .]]]])

Básicamente un acceso directo para abrir una URL y evaluar código contra el medio ambiente DOM remoto:

```
casper.start ( 'Http://google.fr' ).entonces( función () {
    esta . eco( "Estoy en tu Google." );
});

casper.thenOpenAndEvaluate ( 'Http://yahoo.fr' , función () {
    var f = documento .querySelector ( 'formar' );
    f.querySelector ( 'De entrada [name = q]' ).valor = 'Chuck norris' ;
    f.submit ();
});

casper.run ( función () {
    esta . debugPage ();
    esta . salida(); });
});
```

Encadenar()

Firma: Encadenar()

Nuevo en la versión 1.0.

Devuelve una representación de cadena de instancia actual de Casper:

```
casper.start ( 'Http://google.fr' , función () {
    esta . eco( esta ); // [ oponerse Casper], actualmente en http://google.fr/
});

casper.run();
```

unwait ()

Firma: unwait ()

Nuevo en la versión 1.1.

Abortar todos los procesos en espera actuales, si los hubiere.

agente de usuario()

Firma: userAgent (agente String)

Nuevo en la versión 1.0.

establece el `cadena User-Agent` para enviar a través de cabeceras al realizar solicitudes:

```
casper.start();  
  
casper.userAgent ('Mozilla / 5.0 (Macintosh; Intel Mac OS X)';  
  
casper.thenOpen ('Http://google.com/' , función () {  
    esta . eco("Soy un Mac");  
    esta . agente de usuario('Mozilla / 4.0 (compatible; MSIE 6.0; Windows NT 5.1)');  
});  
  
casper.thenOpen ('Http://google.com/' , función () {  
    esta . eco("Soy un PC");  
});  
  
casper.run();
```

`ventana gráfica ()`

Firma: `ventana gráfica (anchura Número, altura Number [, función entonces])`

Cambia el tamaño de ventana gráfica actual:

```
casper.viewport ( 1024 , 768 );
```

Para estar seguro de si la página reído ha ocurrido, hay que utilizarlo de forma asíncrona:

```
casper.viewport ( 1024 , 768 ).entonces( función () {  
    // nuevo puerto vista es ahora efectiva  
});
```

Nuevo en la versión 1.1. A partir del 1.1 puede pasar una `entonces` función de paso directamente a ventana gráfica

`():`

```
casper.viewport ( 1024 , 768 , función () {  
    // nuevo puerto vista es efectiva  
});
```

Nota: PhantomJS viene con un tamaño de visualización predeterminado de 400x300, y CasperJS no anula por defecto.

`visible()`

Firma: `visible (selector String)`

Comprueba si el elemento DOM que concuerden con la presentó `selector de expresión` es visible en la página remota:

```
casper.start ('Http://google.com/' , función () {  
    Si ( esta . visible('#hplogo') ) {  
        esta . eco("Puedo ver el logotipo");  
    } más {  
        esta . eco("No puedo ver el logotipo");  
    }});
```

Espere()

Firma: esperar (Número de tiempo de espera [, a continuación, Función])

Pausa pasos de ejecución suite para una cantidad dada de tiempo, y, opcionalmente, ejecutar un paso en hecho:

```
casper.start ('Http://yoursite.tld', función () {
    esta . Espera( 1000 , función () {
        esta . eco("He esperado por un segundo.");
    });
});

casper.run();
```

También puede escribir la misma cosa como esta:

```
casper.start ('Http://yoursite.tld');

casper.wait( 1000 , función () {
    esta . eco("He esperado por un segundo.");
});

casper.run();
```

waitFor()

Firma: waitFor (Función testFx [, a continuación, la función, la función onTimeout, Número tiempo de espera, información de objeto])

Espera hasta que una función devuelve verdadero para procesar cualquier paso siguiente. También puede establecer un tiempo de espera de devolución de llamada en el uso de la onTimeout argumento, y establecer el tiempo de espera mediante el se acabó el tiempo uno, en milisegundos. El tiempo de espera predeterminado se establece en 5000 ms:

```
casper.start ('Http://yoursite.tld');

casper.waitFor( función comprobar () {
    devolver este . evaluar( función () {
        regreso documento .querySelectorAll ('Ul.your lista li').longitud > 2 ;}); }, función entonces() {

    esta . captureSelector ('Yoursitelist.png' , 'Ul.your-list');
});

casper.run();
```

Ejemplo usando el onTimeout llamar de vuelta:

```
casper.start ('Http://yoursite.tld');

casper.waitFor( función comprobar () {
    devolver este . evaluar( función () {
        regreso documento .querySelectorAll ('Ul.your lista li').longitud > 2 ;}); }, función entonces() {

    // paso a ejecutar cuando verificación () no está mal
    esta . captureSelector ('Yoursitelist.png' , 'Ul.your-list');
}, función se acabó el tiempo() { // paso para ejecutar si de control ha fallado
    esta . eco("No puedo haz mi pantalla." ).salida();
});
```

```
});  
  
casper.run();
```

detalles es una bolsa de propiedades de diversa información que se pasará a la `waitFor.timeout` caso, si se emite. Esto se puede utilizar para obtener mejores mensajes de error o hacer caso omiso de forma condicional algunos eventos de tiempo de espera.

Nota: Todas `waitFor` métodos no son chainable. Considere envolver cada uno de ellos en una `casper.then` con el fin de lograr esto funcionalidad.

Nuevo en la versión 1.1.5.

A partir de 1.1.5 hace una última ejecución de la función de verificación después de tiempo de espera si la función de verificación sigue siendo falsa.

`waitForAlert ()`

Firma: `waitForAlert (Función luego [, función onTimeout, Número de tiempo de espera])`

Nuevo en la versión 1.1-beta4. Espera hasta que una `alerta de JavaScript` se dispara. La función de paso se pasará el mensaje de alerta en el `response.data`

propiedad:

```
casper.waitForAlert ( función ( respuesta) {  
    esta . eco( "Alerta recibida:" + response.data);});
```

`waitForExec ()`

Firma: `waitForExec (comandos, parámetros [, a continuación, la función, la función onTimeout, se acabó el tiempo])`

Nuevo en la versión 1.1.5. Espera hasta que el comando corre con parámetros y salidas. Los de comandos, parámetros, pid, stdout, stderr, elapsedTime y exitCode estarán en el `response.data` propiedad. mando debe ser una cadena o

parámetros debe ser una matriz. mando puede ser una cadena de un ejecutable o una cadena de un ejecutable y sus argumentos separados por espacios. Si mando es Falsy o no es una cadena, shell del sistema (entorno SHELL variable o Comspec) se utiliza. Los argumentos separados por espacios se concatenan con la parámetros matriz que se envía al ejecutable. Si parámetros es Falsy o no es una matriz, se utiliza una matriz vacía. se acabó el tiempo pueden ser un número o un conjunto de dos números, el primero es el tiempo de espera de `Espere*` funciones de la familia y el segundo es el tiempo de espera entre TERM y matar señales en tiempo de espera. Si no se declara, asume el mismo valor del primer elemento o el tiempo de espera predeterminado de

`Espere*` funciones de la familia ..

```
// combinación PDFs capturados con shell del sistema por defecto (Bash en Linux) llamando /usr/bin/gs,  
-- y se ejecuta un pequeño script para eliminar archivos  
casper.waitForExec ( nulo , [ '-d', '/usr/bin/gs -dPDFSETTINGS = /ebook -dBATCH -dNOPAUSE -  
-q -sDEVICE = pdfwrite -sOutputFile = /my_merged_captures.pdf /my_captures*.pdf && /bin /  
./my_captures*.pdf rm; } || { /bin /rm /my_merged_captures.pdf && exit 1; } ],  
función ( respuesta) {  
    esta . eco( "Proceso finalizado por sí mismo:" + JSON.stringify (response.data)); }, función ( tiempo de espera, la respuesta) {  
  
    esta . eco( "Programa terminado por Casper:" + JSON.stringify (response.data));  
});
```

```
// combinación capturado archivos PDF con llamadas bash / usr / bin / gs, y ejecuta un pequeño script para
// eliminar archivos
casper.waitForExec ('/ Bin / bash -c ', ['!/ Usr / bin / gs -dPDFSETTINGS = / ebook -dBATCH -
.. dNOPAUSE q = -sDEVICE pdfwrite -sOutputFile = / my_merged_captures.pdf / my_captures * .
.. /my_captures*.pdf pdf & / bin / rm; } || {/ Bin / rm /my_merged_captures.pdf && exit 1;
.. /};

función ( respuesta) {
    esta . eco( "Proceso finalizado por sí mismo:" + JSON.stringify (response.data)); }, función ( tiempo de espera, la respuesta) {

    esta . eco( "Programa terminado por Casper:" + JSON.stringify (response.data));
});

// combinación de archivos PDF capturado llamando gs / usr / bin /
casper.waitForExec ('/ usr / bin / gs', ['-dPDFSETTINGS = / ebook', '-dBATCH', '-dNOPAUSE', '-q', '-
.. sDEVICE = pdfwrite', '-sOutputFile = / my_merged_captures_pdfs.pdf', '/my_captures_1.pdf',
.. ,/my_captures_2.pdf', '/my_captures_3.pdf'],
función ( respuesta) {
    esta . eco( "Proceso finalizado por sí mismo:" + JSON.stringify (response.data)); }, función ( tiempo de espera, la respuesta) {

    esta . eco( "Programa terminado por Casper:" + JSON.stringify (response.data));
});

// combinación de archivos PDF capturado llamando gs / usr / bin /
casper.waitForExec ('/ Usr / bin / gs -dPDFSETTINGS = / ebook -dBATCH -dNOPAUSE -q -
.. sDEVICE = pdfwrite -sOutputFile = / my_merged_captures_pdfs.pdf /my_captures_1.pdf / MY_
.. captures_2.pdf /my_captures_3.pdf', nulo ,
función ( respuesta) {
    esta . eco( "Proceso finalizado por sí mismo:" + JSON.stringify (response.data)); }, función ( tiempo de espera, la respuesta) {

    esta . eco( "Programa terminado por Casper:" + JSON.stringify (response.data));
});
```

Nota: waitForExec () sólo mata a la llamada del programa en tiempo de espera. Si el programa que se llama llama a otros procesos, no van a ser asesinados cuando waitForExec () tiempo de espera.

waitForPopup ()

Firma: waitForPopup (String | RegExp | urlPattern Object [, a continuación, la función, la función onTimeout, Número de tiempo de espera])

Nuevo en la versión 1.0.

Espera a una ventana emergente que tiene su url coincide con el patrón proporcionado para ser abierta y cargado. Las ventanas emergentes

cargados actualmente están disponibles en el Casper.popups matriz similar a la propiedad:

```
casper.start ('Http://foo.bar').entonces( función () {
    esta . test.assertTitle ( 'Título principal página' );
    esta . clickLabel ( 'Ábreme una ventana emergente' );
});

// esto esperará a que el emergente para ser abierto y cargado
casper.waitForPopup ( /popup.html$/ , función () {
    esta . (test.assertEquals esta . popups.length, 1 );
});
```

```
// esto esperar a que la primera ventana emergente para ser abierto y cargado
casper.waitForPopup ( 0 , función () {
    esta . (test.assertEquals esta . popups.length, 1 );
});

// esto esperará a que el emergente de llamada para ser abierto y cargado
casper.waitForPopup ({windowName : "MainPopup" }, función () {
    esta . (test.assertEquals esta . popups.length, 1 );
});

// esto va a esperar por el título de la ventana emergente para ser abierto y cargado
casper.waitForPopup ({title : "Emergente TITLE" }, función () {
    esta . (test.assertEquals esta . popups.length, 1 );
});

// esto esperará url de la ventana emergente para ser abierto y cargado
casper.waitForPopup ({url : 'Http://foo.bar/' }, función () {
    esta . (test.assertEquals esta . popups.length, 1 );
});

// esto hará que el DOM emergente como el principal activo sólo por el tiempo que se ejecuta el cierre de paso //

casper.withPopup (/popup\.html$/ , función () {
    esta . test.assertTitle ('Título emergente');
});

// siguiente paso pasa automáticamente la página actual a la inicial
casper.then ( función () {
    esta . test.assertTitle ('Título principal página');
});
```

waitForResource ()

Firma: `waitForResource (String | Función | RegExp testFx [, a continuación, la función, Función onTimeout, Número de tiempo de espera])`

Espera hasta que un recurso que coincide con un limitaciones de recursos que coinciden definido por testFx se satisfacen para procesar un siguiente paso. Los testFx argumento puede ser una cadena, una función o una RegExp ejemplo:

```
casper.waitForResource ( "Foobar.png" , función () {
    esta . eco( 'Foobar.png se ha cargado.' );
});
```

Usando una expresión regular:

```
casper.waitForResource ( /foo(bar|baz)\.png$/ , función () {
    esta . eco( 'Foobar.png o foobaz.png ha sido cargado.' );
});
```

El uso de una función:

```
casper.waitForResource ( función testResource (recurso) {
    regreso resource.url.indexOf ( "Https" ) === 0 ; }, función onReceived () {
```

```
    esta . eco( 'Un recurso seguro se ha cargado. );
});
```

waitForUrl ()

Firma: waitForUrl (String | RegExp url [, a continuación, la función, la función onTimeout, tiempo de espera número])

Nuevo en la versión 1.1.

Espera a que la URL de la página actual para que coincida con el argumento proporcionado (Cuerda o RegExp):

```
casper.start( 'Http://foo/' ) .waitForUrl( '/login.html$/ , función () {
    esta . eco( 'Redirigido a login.html' );
});

casper.run();
```

waitForSelector ()

Firma: waitForSelector (selector String [, a continuación, la función, la función onTimeout, tiempo de espera número])

Espera hasta que un elemento que coinciden con el proporcionado *selector de expresión* existe en DOM a distancia para procesar cualquier paso siguiente. Usos *waitFor ()* :

```
casper.start( 'Https://twitter.com/#!/n1k0' );

casper.waitForSelector( '.tweet fila' , función () {
    esta . captureSelector( 'Twitter.png' , 'Html' );
});

casper.run();
```

waitWhileSelector ()

Firma: waitWhileSelector (selector String [, a continuación, la función, la función onTimeout, Número de tiempo de espera])

Espera hasta que un elemento que coinciden con el proporcionado *selector de expresión* no existe en DOM a distancia para procesar un siguiente paso. Usos *waitFor ()* :

```
casper.start( 'Http://foo.bar/' );

casper.waitWhileSelector( '.selector' , función () {
    esta . eco( '.Selector ya no existe!' );
});

casper.run();
```

waitForSelectorTextChange ()

Firma: waitForSelectorTextChange (selectores String [, función de entonces, Función onTimeout, Número de tiempo de espera])

Espera hasta que el texto en un elemento que coincide con el proporcionado *selector de expresión* se cambia a un valor diferente antes de procesar el siguiente paso.

Usos *waitFor()*:

```
casper.start ('Http://foo.bar');

casper.waitForSelectorTextChange ('.selector' , función () {
    esta . eco( 'El texto en .Selector ha cambiado.' );
});

casper.run();
```

waitForText ()

Firma: waitForText (texto String [, a continuación, la función, la función onTimeout, Número se acabó el tiempo])

Nuevo en la versión 1.0.

Espera hasta que el texto aprobado está presente en los contenidos de la página antes de procesar el siguiente paso de inmediato. Usos *waitFor()*:

```
casper.start ('Http://why.univer.se') .waitForText ( "42" , función () {
    esta . eco( Encontrado la respuesta. ' );
};

casper.run();
```

waitUntilVisible ()

Firma: waitUntilVisible (selector String [, a continuación, la función, la función onTimeout, Número de tiempo de espera])

Espera hasta que un elemento que coinciden con el proporcionado *selector de expresión* es visible en el DOM remoto para procesar un siguiente paso. Usos *waitFor()*.

waitWhileVisible ()

Firma: waitWhileVisible (selector String [, a continuación, la función, la función onTimeout, Número de tiempo de espera])

Espera hasta que un elemento que coinciden con el proporcionado *selector de expresión* ya no es visible en DOM a distancia para procesar un siguiente paso. Usos *waitFor()*:

```
var Casper = exigir ( 'Casper' ).crear();

casper.start ('Https://www.example.com') .thenClick ( 'Html div cuerpo pa' , función () {
    esta . waitWhileVisible ( 'Body> div: nth-child (1)> p: nth-child (2)' , función () {
        esta . eco( "El elemento seleccionado existía en la página anterior, pero no existe en
        .. esta página." ); })
}).correr();
```

advertir()

Firma: advertir (mensaje String)

Registros y muestra un mensaje de advertencia a la salida estándar:

```
casper.warn ( "Soy un mensaje de advertencia." );
```

Nota: Vocación **advertir()** dará lugar a la advertir *evento*.

withFrame ()

Firma: withFrame (String | Número frameInfo, Función continuación)

Nuevo en la versión 1.0.

Cambia la página principal para la trama que tiene el nombre o el número de trama de adaptación de índices el argumento pasado, y procesa un paso.

El cambio de contexto página sólo dura hasta que la ejecución paso es acabada:

```
casper.start ( 'Pruebas / site / frames.html' , función () {
    esta . test.assertTitle ( 'FRAMESET TÍTULO' );
});

casper.withFrame ( 'Frame1' , función () {
    esta . test.assertTitle ( 'Marco de título' );
});

casper.withFrame ( 0 , función () {
    esta . test.assertTitle ( 'Marco de título' );
});

casper.then ( función () {
    esta . test.assertTitle ( 'FRAMESET TÍTULO' );
});
```

withPopup ()

Firma: withPopup (Mezclado popupInfo, Función continuación)

Nuevo en la versión 1.0.

Cambia la página principal a una ventana emergente cotejar la información de pasado como parámetro, y procesa un paso. El cambio de contexto página sólo dura hasta que la ejecución paso es acabada:

```
casper.start ( 'Http://foo.bar' ).entonces( función () {
    esta . test.assertTitle ( 'Título principal página' );
    esta . clickLabel ( 'Abre una ventana emergente' );
});

// esto esperará a que el emergente para ser abierto y cargado
casper.waitForPopup ( /popup.html$/ , función () {
    esta . (test.assertEquals esta . popups.length, 1 );
});
```

```
// esto hará que el DOM emergente como el principal activo sólo por el tiempo que se ejecuta el cierre de paso //

casper.withPopup ( /popup\.html$/ , función () {
    esta . test.assertTitle ( 'Título emergente' );
});

// esto hará que el DOM emergente como el principal activo sólo por el tiempo que se ejecuta el cierre de paso //

casper.withPopup ( 0 , función () {
    esta . test.assertTitle ( 'Título emergente' );
});

// esto hará que el DOM emergente como el principal activo sólo por el tiempo que se ejecuta el cierre de paso //

casper.withPopup ({windowName : "MainPopup" , título : 'Título emergente' , url : 'Http://foo.bar/' })
.. función () {
    esta . test.assertTitle ( 'Título emergente' );
};

// siguiente paso pasa automáticamente la página actual a la inicial
casper.then ( función () {
    esta . test.assertTitle ( 'Título principal página' );
});
```

Nota: Las ventanas emergentes cargados actualmente están disponibles en el Casper.popups matriz similar a la propiedad.

withSelectorScope ()

Firma: withSelectorScope (selector de cuerdas, a continuación, la función)

Nuevo en la versión 1.1.5.

Cambia el ámbito principal DOM a un ámbito si especifica la información pasa como argumento, y procesa un paso. El cambio de contexto alcance sólo dura hasta la ejecución paso es terminado:

```
.. índice :: Enfocar
```

enfocar()

Firma: zoom (factor de Número)

Nuevo en la versión 1.0.

Establece el factor de zoom de la página actual:

```
var Casper = exigir( 'Casper' ).crear();

casper.start () . zoom ( 2 ) . thenOpen ( 'Http://google.com' , función () {
    esta . capturar( 'Big-google.png' );
});

casper.run();
```

los clientutils módulo

barcos Casper con unas pocas utilidades del lado del cliente que se inyectan en el entorno de DOM remoto, y accesibles desde allí a través de la `__utils__` instancia de objeto de la ClientUtils clase del clientutils módulo:

```
casper.evaluate ( función () {
    __utils__.echo ( "¡Hola Mundo!" );
});
```

Nota: Estas herramientas se proporcionan para evitar CasperJS de acoplamiento a cualquier biblioteca de terceros, como jQuery, Mootools o algo; pero siempre puede incluir éstos y tenerlos disponibles en el cliente utilizando el `Casper.options.clientScripts` opción.

bookmarklet

Un bookmarklet también está disponible para ayudar a la inyección de utilidades del lado del cliente de Casper en el DOM de su navegador favorito. Sólo tienes que arrastrar el siguiente enlace a tus favoritos barra de herramientas de; al hacer clic en él, un `__utils__` objeto estará disponible dentro de la consola de su navegador:

Nota: CasperJS y PhantomJS se basan en Webkit , Se le anima encarecidamente el uso de una reciente navegador compatible con Webkit para utilizar este bookmarklet (Chrome, Safari, etc.).

ClientUtils prototipo

echo()

Firma: echo (mensaje String)

Nuevo en la versión 1.0.

Imprimir un mensaje a la consola Casper desde el entorno de la página DOM remoto:

```
casper.start ( 'Http://foo.net/' ).thenEvaluate ( función () {
    __utils__.echo ( 'plat' ); // esto será impreso en su cáscara en tiempo de ejecución
});
```

codificar()

Firma: encode (contenido de la cadena)

Codifica una cadena utilizando la algoritmo de base 64 . Para los registros, CasperJS no utiliza incorporado `window.btoa()` función, ya que no se puede tratar el cliente con las secuencias codificadas usando > 8b caracteres:

```
var base64; casper.start ( 'Http://foo.bar' , función () {

    base64 = esta . evaluar( función () {
        regreso __utils__.codificar ( "Yo he sido un poco críptica recientemente" );
    });
});
```

```
casper.run ( función () {
    esta . echo (base64) .Salir ();});
```

existe ()

Firma: existe (selector de cadena)

Comprueba si un elemento DOM búsqueda de un hecho *selector de expresión* existe:

```
var existe; casper.start ( 'Http://foo.bar' , función () {

    existe = esta . evaluar( función () {
        regreso __ utils __ existe ( '#some_id' );
   }); });

casper.run ( función () {
    esta . echo (existe) .Salir ();});
```

Encuentra todos()

Firma: findAll (selector de cadena)

Recupera todos los elementos DOM que coinciden con un dato *selector de expresión*:

```
var campo de golf; casper.start ( 'Http://foo.bar' , función () {

    campo de golf = esta . evaluar( función () {
        var elementos = __utils __ .findAll ( 'un menú' );
        regreso elementos.map ( función ( e){
            regreso e.getAttribute ( 'Href' );
       }); });
    });

casper.run ( función () {
    esta . eco (JSON.stringify (links)) exit ();});
```

Encuentra uno()

Firma: findOne (selector de cadena)

Recupera un solo elemento DOM por una *selector de expresión*:

```
var href; casper.start ( 'Http://foo.bar' , función () {

    href = esta . evaluar( función () {
        regreso __ utils __ findOne ( '#mi identificación' ) .getAttribute ( 'Href' );
   }); });
```

```
casper.run ( función () {
    esta . echo (href) .Salir ();});
```

forceTarget ()

Firma: forceTarget (selector String, target String)

Forzar el motor a utilizar otro destino en lugar de la prevista. Muy útil para limitar el número de ventanas abiertas y reducir el consumo de memoria:

```
casper.start ( 'Http://foo.bar' , función () {
    var href = esta . evaluar( función () {
        regreso __ utils __ forceTarget ( '#mi identificación' , '_yo' ).hacer clic();
    });
    esta . echo (href); });

casper.run ( función () {
    esta . salida();});
```

getBase64 ()

Firma: getBase64 (String url [, método String, datos de objeto])

Este método será recuperada una versión codificada en base64 de cualquier recurso detrás de una URL. Por ejemplo, imaginemos que queremos recuperar la representación base 64 del logotipo de algún sitio web:

```
var logo = nulo ;
casper.start ( 'Http://foo.bar' , función () {
    logo = esta . evaluar( función () {
        var imgURL = documento .querySelector ( 'Img.logo' ) .getAttribute ( 'Origen' );
        regreso __ . Utils __ getBase64 (imgURL);});});

casper.run ( función () {
    esta . eco (logotipo) .Salir ();});
```

getBinary ()

Firma: getBinary (String url [, método String, datos de objeto])

Este método será recuperado el contenido en bruto de un recurso binario dado; Lamentablemente, sin embargo, PhantomJS no puede procesar estos datos directamente de modo que tendrá que procesarlos dentro del entorno DOM remoto. Si la intención de descargar el recurso, el uso [getBase64 \(\)](#) o [Casper.base64encode \(\)](#) en lugar:

```
casper.start ( 'Http://foo.bar' , función () {
    esta . evaluar( función () {
        var imgURL = documento .querySelector ( 'Img.logo' ) .getAttribute ( 'Origen' ); console.log ( __ __ utils getBinary (imgURL).);});});
```

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

```
casper.run();
```

getDocumentHeight ()

Firma: getDocumentHeight ()

Nuevo en la versión 1.0.

Recupera altura del documento actual:

```
var documentHeight;

casper.start('Http://google.com/', función () {
    documentHeight = esta . evaluar( función () {
        regreso __ utils __ getDocumentHeight () .; });

    esta . eco( 'Altura del documento es' + documentHeight + 'Px' );
});

casper.run();
```

getDocumentWidth ()

Firma: getDocumentWidth ()

Nuevo en la versión 1.0.

Recupera anchura documento actual:

```
var documentHeight;

casper.start('Http://google.com/', función () {
    documentWidth = esta . evaluar( función () {
        regreso __ utils __ getDocumentWidth () .; });

    esta . eco( 'Anchura del documento es' + documentWidth + 'Px' );
});

casper.run();
```

getElementBounds ()

Firma: getElementBounds (selector de cadena)

Recupera límites por unos elementos DOM que concuerden con la proporcionada *selector*.

Devuelve un objeto con cuatro teclas: **superior, izquierda, ancho y altura**, o nulo si no existe el selector.

getElementsBounds ()

Firma: getElementsBounds (selector de cadena)

Recupera límites para todos elemento DOM que coincide con el previsto *selector*.

Devuelve una matriz de objetos que tienen cada uno cuatro teclas: superior, izquierda, ancho y altura.

getElementByXPath ()

Firma: getElementByXPath (expresión String [, el alcance HTMLElement])

Recupera un solo elemento DOM búsqueda de un dado *expresión XPath*.

Nuevo en la versión 1.0. los alcance argumento permiten establecer el contexto de la ejecución de la consulta XPath:

```
// se llevará a cabo en contra de todo el documento
__utils __. getElementByXPath ( '//un' );

// se llevará a cabo contra un elemento de DOM dado
__utils __. getElementByXPath ( '//un' , __utils __. FindOne ( 'Div.main' ));
```

getElementsByXPath ()

Firma: getElementsByXPath (expresión String [, el alcance HTMLElement])

Recupera todos los elementos DOM que coinciden con un dado *expresión XPath*, si los hubiere. Nuevo en la versión 1.0.

los alcance argumento permite establecer el contexto de la ejecución de la consulta XPath.

GetFieldValue ()

Firma: GetFieldValue (selector String [, el alcance HTMLElement])

Nuevo en la versión 1.0.

Recupera el valor del campo nombrado en contra de la inputNamed argumento:

```
< formar >
    < entrada tipo = "Texto" nombre = valor "plop" = "42" >
</ formar >
```

Utilizando el GetFieldValue () método para plaf:

```
__utils __. GetFieldValue ( "[Name = "plop"]" ); // 42
```

opciones:

getFormValues ()

Firma: getFormValues (selector de cadena)

Nuevo en la versión 1.0.

Recupera una forma dada y todos sus valores de campo:

```
<formar camé de identidad = acción "login" = "/iniciar sesión" >< entrada tipo = "Texto" nombre = valor
  "nombre de usuario" = "Foo" >< entrada tipo = "Texto" nombre = valor "contraseña" = "bar" >< entrada
  tipo = "enviar" >

</ formar >
```

Para obtener los valores de la forma:

```
_utils __. getFormValues ( 'Forma # login' ); // { Nombre de usuario: 'foo', contraseña: 'bar' }
```

Iniciar sesión()

Firma: log (mensaje String [, nivel String])

Registra un mensaje con un nivel opcional. Formateará el mensaje una manera CasperJS podrá conectarse lado PhantomJS. El nivel predeterminado es depurar:

```
casper.start ( 'Http://foo.net' ) .thenEvaluate ( función () {
  __utils __. log ( "Tenemos un problema en el lado del cliente" , 'error' );
});
```

makeSelector ()

Firma: makeSelector (selector String [, tipo String])

Nuevo en la versión 1.1-beta5.

Hace selector de tipo de fi nida XPath, nombre o etiqueta. Tiene la función mismo resultado que en el módulo selectXPath Casper para el tipo XPath -, se hace objeto XPath. Función también acepta el nombre del atributo del campo forma fi o puede seleccionar elemento por su texto de la etiqueta.

Parámetro tipo valores:

- 'Css'

selector CSS3 - selector se volvió transparente

- 'XPath' || nulo

selector XPath - retorno objeto XPath

- 'Nombre' || 'nombres'

seleccionar la entrada del nombre específico, tanto interna encubierta al selector CSS3

- 'Etiqueta' || 'etiquetas'

seleccionar la entrada de la etiqueta específica, internamente convertida en selector XPath. Como selector es texto utilizado Ejemplos de etiquetas:

```
_utils __. makeSelector ( '// li [texto () = "bla"]' , 'XPath' ); // {volver tipo: 'XPath',
  ruta: '// li [texto () = "bla"]'
}
_utils __. makeSelector ( 'parámetro' , 'nombre' ); // retorno [name = "parámetro"]
_utils __. makeSelector ( 'Mi etiqueta' , 'etiqueta' ); // {volver tipo: 'XPath', ruta: '//',
  id = string (// etiqueta [Texto () = "Mi etiqueta"] / @ para) }
```

mouseEvent ()

Firma: mouseEvent (tipo String, selector de cadena, [Número | Cadena X, Número | Cadena Y])

Distribuye un evento del ratón en el elemento DOM detrás del selector proporcionado. eventos son compatibles mouseup, mousedown, clic, dblclick, mousemove, al pasar el ratón, mouseout, MouseEnter, mouseleave y Menú de contexto:

```
... índice :: XPath
```

removeElementsByXPath ()

Firma: removeElementsByXPath (expresión String)

Elimina todos los elementos del DOM que coinciden con un dado *expresión XPath*.

sendAJAX ()

Firma: sendAJAX (String url [, método String, los datos de objeto, async Boolean, la configuración del objeto])

Nuevo en la versión 1.0.

Envía una petición AJAX, utilizando los siguientes parámetros:

- url: La URL para solicitar.
- método: El método HTTP (por defecto: OBTENER).
- datos: parámetros de la petición (por defecto: nulo).
- asíncrono: Bandera para una solicitud de asynchronous? (defecto: falso)
- ajustes: Cabeceras personalizadas cuando realizan la petición AJAX (por defecto: nulo). ADVERTENCIA: una cabecera inválida aquí puede hacer la petición falla en silencio.

Advertencia: No se olvide de pasar el - web-security = sin opción en su llamada de CLI para realizar dominios cruzados solicita cuando sea necesario:

```
var datos, wsurl = 'Http://api.site.com/search.json';

casper.start ('Http://my.site.com', función () {
    datos = esta . evaluar( función ( wsurl) {
        regreso JSON.parse (____ utilidades. SendAJAX (wsurl, 'OBTENER' , nulo , falso ));
    }, {Wsurl : wsurl}); });

casper.then ( función () {
    exigir('utils') .dump (datos);
});
```

SetFieldValue ()

Firma: SetFieldValue (String | Selector de objetos, el valor Mixta [, HTMLElement alcance])

Nuevo en la versión 1.1-beta5.

Establece un valor para formar campo de CSS3 o el selector XPath. Con [makeSelector \(\)](#) función se puede utilizar fácilmente con nombre o etiqueta selector

opciones

- (String | Object) alcance: Selector:

específicas Forma C Ejemplos

alcance:

```
__utils__.SetFieldValue( "Entrada [name = 'email']" , 'Chuck@norris.com' ); __utils__.SetFieldValue( "Entrada [name = 'email']" , 'Chuck@norris.com' , { 'FormSelector' : 'myform' });
__utils__.SetFieldValue( __utils__.makeSelector( 'correo electrónico' , 'nombre' ), 'Chuck@norris.com' );
```

visible()

Firma: visible (selector String)

Comprueba si un elemento es visible:

```
var logIsVisible = casper.evaluate( función () {
    regreso __utils__.visible( 'H1' );
});
```

los colorizer módulo

los colorizer módulo contiene una colorizer clase que puede generar cadenas ANSI de colores:

```
var colorizer = exigir( 'Colorizer' ).crear( 'Colorizer' ); console.log( colorizer.colorize( "Hola Mundo" , "INFO" ));
```

Aunque la mayoría de las veces lo va a usar de forma transparente mediante el [Casper.echo \(\)](#) método:

```
casper.echo( 'Un mensaje informativo' , 'INFO' ); // impreso en verde
casper.echo( 'Un mensaje de error' , 'ERROR' ); // impreso en rojo
```

Saltarse las operaciones de peinado CasperJS

Si desea saltar toda la operación de coloración y obtener sin color de texto sin formato, acaba de establecer la colorizerType Casper opción a Tonto:

```
var Casper = exigir('Casper').Create({colorizerType: 'Tonto'});

casper.echo("Hola", "INFO");
```

Nota: Esto es especialmente útil si usted está utilizando CasperJS en la plataforma Windows, ya que no hay soporte para la salida de color en esta plataforma.

estilos disponibles prede fi nido

Disponible prede fi nidas estilos son:

- ERROR: texto blanco sobre fondo rojo
 - INFORMACIÓN: texto verde
 - RASTRO: texto verde
 - PARÁMETRO: texto cian
 - COMENTARIO: texto amarillo
 - ADVERTENCIA: texto rojo
 - GREEN_BAR: texto blanco sobre fondo verde
 - Red_bar: texto blanco sobre fondo rojo
 - INFO_BAR: texto cian
 - WARN_BAR: texto blanco sobre fondo naranja
- He aquí un ejemplo

del resultado de lo que puede verse como:

colorear ()

Firma: colorear (texto String, String styleName)

Calcula una versión en color de la cadena de texto que se proporciona el uso de un determinado estilo prede fi nida:

```
var colorizer = exigir('Colorizer').crear(); console.log(colorizer.colorize("Soy un rojo de error", "ERROR"));
```

Nota: La mayoría de las veces no tendrá que utilizar una colorizer ejemplo directamente como CasperJS proporciona todos los métodos necesarios.

Ver la lista de la [estilos prede fi nido disponibles](#).

formato()

Firma: formato (texto de cuerdas, estilo de objeto)

Da formato a una cadena de texto usando el estilo proporcionado definición. Una definición de estilo de fi es una norma javascript Objeto ejemplo que puede de definir las siguientes propiedades:

```

niko@NikoBook:~/Sites/casperjs$ phantomjs tests/run.js
# logging GitHub, Inc. [US] https://github.com/n1k0/casperjs
PASS log() adds a log entry
# navigating bookmarklets utils python design mercurial git django gottle
PASS start() can add a new navigation step Casper ships with a few client-side utilities which
PASS then() adds a new navigation step instance of the phantom.Casper.ClientUtils c
PASS start() casper can start itself an open an url
PASS start() injects ClientUtils instance within remote DOM getBase64(String url)
# encoding
PASS base64encode() can retrieve base64 contents This method will retrieved a base64 encoded versi
representation of some website's logo:
# clicking
PASS click() casper can click on a text link and react when it is loaded 1/2
PASS click() casper can click on a text link and react when it is loaded 2/2
# filling a form
PASS fill() can fill an input[type=text] form field casper.start('http://100bar.com', function()
PASS fill() can fill a textarea form field logo = self.evaluate(function() {
PASS fill() can pick a value from a select form field });
PASS fill() can check a form checkbox var imgUrl = document.querySelector('img');
PASS fill() can check a form radio button 1/2 self.echo(logo).exit();
PASS fill() can check a form radio button 2/2 });
PASS fill() can select a file to upload
PASS click() casper can click on a submit button
PASS fill() input[type=email] field was submitted Colorizer
PASS fill() textarea field was submitted
PASS fill() input[type=checkbox] field was submitted ships with a Colorizer object which can
PASS fill() input[type=radio] field was submitted
PASS fill() select field was submitted casper.echo('this is an informative message');
PASS log() logged messages casper.echo('this is an error message');
PASS 22 tests executed, 22 passed, 0 failed.
Available predefined styles are:
niko@NikoBook:~/Sites/casperjs$ 

```

- Cuerda BG: Nombre del color de fondo
- Cuerda FG: Nombre del color de primer plano
- Boole negrita: aplicar el formato de negrita
- Boole guion bajo: aplicar formato de subrayado
- Boole parpadeo: aplicar el formato de abrir y cerrar
- Boole marcha atrás: aplicar formato inversa
- Boole encubrir: aplicar ocultar formato

Nota: nombres de los colores disponibles son negro, rojo, verde, amarillo, azul, magenta, cian y blanco:

```
var colorizer = exigir( 'Colorizer' ).crear(); colorizer.format ( "Todos vivimos en un submarino amarillo" , {Bg :  
'amarillo' , fg : 'azul' , negrita : cierto  
});
```

los ratón módulo

los Ratón clase

los Ratón clase es una abstracción en la parte superior de diversas operaciones del ratón como en movimiento, clic, doble clic, vuelcos, etc. Se requiere una Casper instancia como una dependencia para acceder al DOM. Un objeto del ratón se puede crear de esa manera:

```
var Casper = exigir( "Casper" ).crear();  
var ratón = exigir( "ratón" ) .Create (Casper);
```

Nota: UN Casper instancia tiene una ratón la propiedad ya se ha definido, por lo que normalmente no tiene que crear uno por su lado en los scripts de Casper:

```
casper.then ( función () {  
    esta . click del ratón( 400 , 300 ); // hace clic en las coordenadas x = 400; y = 300  
});
```

hacer clic()

Firma:

- clic (número x, Número y)
- clic (selector de cadena)
- clic (selector de cadena, un número X, Número y)

Realiza un clic en el primer elemento que se ajusten a la prevista *selector de expresión* o en las coordenadas dadas si se pasan dos números:

```
casper.then ( función () {
    esta . click del ratón( "#mi enlace" ); // hace clic <a id="my-link"> bueno </a>
    esta . click del ratón( 400 , 300 ); // hace clic en las coordenadas x = 400; y = 300
});
```

Nota: Es posible que desee utilizar directamente `Casper#clic` en lugar.

haga doble clic()

Firma:

- DoubleClick (Número de x, Número y)
- DoubleClick (selector de cadena)
- DoubleClick (selector String, Número x, Número y)

envía una `haga doble clic` evento del ratón sobre el elemento de búsqueda de los argumentos proporcionados:

```
casper.then ( función () {
    esta . mouse.doubleclick ( "#mi enlace" ); // doubleclicks <a id="my-link"> bueno </a>
    esta . mouse.doubleclick ( 400 , 300 ); // doubleclicks en las coordenadas x = 400; y = 300
});
```

botón derecho del ratón()

Firma:

- rightclick (Número de x, Número y)
- botón derecho del ratón (selector de cadena)
- rightclick (selector String, Número x, Número y)

envía una Menú de contexto evento del ratón sobre el elemento de búsqueda de los argumentos proporcionados:

```
casper.then ( función () {
    esta . mouse.rightclick ( "#mi enlace" ); // doubleclicks <a id="my-link"> bueno </a>
    esta . mouse.rightclick ( 400 , 300 ); // doubleclicks en las coordenadas x = 400; y = 300
});
```

abajo()

Firma:

- hacia abajo (Número x, Número y)
- abajo (selector de cadena)
- abajo (selector de cadena, un número X, Número y)

envía una ratón hacia abajo evento del ratón sobre el elemento de búsqueda de los argumentos proporcionados:

```
casper.then ( función () {
    esta . ratón hacia abajo( "#mi enlace" ); // pulse el botón a la izquierda por <a id="my-link"> bueno </a>
    esta . ratón hacia abajo( 400 , 300 ); // pulse botón izquierdo en las coordenadas x = 400; y = 300
});
```

movimiento()

Firma:

- movimiento (Número de x, Número y)
- movimiento (selector de cadena)
- mover (selector String, Número x, Número y)

Mueve el cursor del ratón sobre el elemento de búsqueda de los argumentos proporcionados:

```
casper.then ( función () {
    esta . mouse.move ( "#mi enlace" ); // mueve el cursor sobre <a id="my-link"> bueno </a>
    esta . mouse.move ( 400 , 300 ); // mueve cursor sobre las coordenadas x = 400; y = 300
});
```

arriba()

Firma:

- hasta (Número de x, Número y)
- arriba (selector de cadena)
- arriba (selector de cadena, un número X, Número y)

envía una mouseup evento del ratón sobre el elemento de búsqueda de los argumentos proporcionados:

```
casper.then ( función () {
    esta . mouse.up ( "#mi enlace" ); // suelte el botón izquierdo sobre <a id="my-link"> bueno </a>
    esta . mouse.up ( 400 , 300 ); // liberar el botón izquierdo sobre las coordenadas x = 400; y = 300
});
```

los ensayador módulo

Casper barcos con una ensayador módulo y una Ensayador la clase que proporciona una API para la unidad y el propósito de prueba funcional. Por defecto se puede acceder a una instancia de esta clase a través de la prueba propiedad de cualquier Casper instancia de clase.

Nota: La mejor manera de aprender cómo utilizar la API de Tester y verlo en acción es, probablemente, a echar un vistazo a propios conjuntos de pruebas CasperJS' .

los Ensayador prototipo

afirmar()

Firma: afirmar (condición Boolean [, mensaje String])

Afirma que la condición facilitada resuelve estrictamente a un valor lógico cierto:

```
casper.test.assert( cierto , "verdad es verdad" ); casper.test.assert( !falso , "verdad  
está ahí fuera" );
```

Ver también:

[assertNot \(\)](#)

[assertDoesntExist \(\)](#)

Firma: assertDoesntExist (selector String [, mensaje String])

Afirma que un elemento que coinciden con el proporcionado *selector de expresión* ¿no existe dentro del entorno DOM remoto:

```
casper.test.begin( 'assertDoesntExist () pruebas' , 1 , función ( prueba ) {  
    casper.start(). entonces ( función () {  
        esta . setContent ( '<Div class = "cielo"> </ div>' ); test.assertDoesntExist ( '.impuestos' );  
  
    }).correr( función () {  
        test.done (); } ); } );
```

Ver también:

[assertExists \(\)](#)

[assertEquals \(\)](#)

Firma: assertEquals (TestValue mezclado, mezclado esperado [, mensaje String])

Afirma que los dos valores son estrictamente equivalentes:

```
casper.test.begin( 'assertEquals () pruebas' , 3 , función ( prueba ) {  
    (test.assertEquals 1 + 1 , 2 ); test.assertEquals ([ 1 , 2 , 3 ], [ 1 , 2 , 3 ]); test.assertEquals ({a : 1 ,  
    b : 2 }, {un : 1 , b : 2 }); test.done (); } );
```

Ver también:

[assertNotEquals \(\)](#)

[assertEval \(\)](#)

Firma: assertEval (función fn [, Mensaje de cadena, los argumentos mixtos])

Afirma que una *Evaluación código en DOM remoto* estrictamente resuelve un booleano cierto:

```
casper.test.begin( 'AssertEval () prueba' , 1 , función ( prueba ) {  
    casper.start(). entonces ( función () {  
        esta . setContent ( '<Div class = "cielo"> cerveza </ div>' ); test.assertEval ( función () {  
  
            regreso __ utils __ findOne ( '.cielo' ).contenido del texto === 'cerveza' ; } );  
    }).correr( función () {  
        test.done (); } ); } );
```

```
}).correr( función () {
    test.done ();});});
```

assertEvalEquals ()

Firma: assertEvalEquals (fn Función, mezclado espera [, String mensaje, Mezclado argumentos])

Afirma que el resultado de una *Evaluación código en DOM remoto* estrictamente igual al valor esperado:

```
casper.test.begin('assertEvalEquals () pruebas', 1, función ( prueba) {
    casper.start().entonces( función () {
        esta .setContent ('<Div class = "cielo"> cerveza </ div>'); (test.assertEvalEquals función () {
            regreso __utils__.findOne('.cielo').contenido del texto;
        }, 'cerveza');
    }).correr( función () {
        test.done ();});});
```

assertCount ()

Firma: assertassertCount (selector de cadena, recuento Número [, String mensaje])

Afirma que una *selector de expresión* coincide con un número dado de elementos:

```
casper.test.begin('assertCount () pruebas', 3, función ( prueba) {
    casper.start().entonces( función () {
        esta . contenido de página = '<Ul> <li> 1 </ li> <li> 2 </ li> <li> 3 </ li> </ ul>'; testassertCount ('Ul', 1);
        testassertCount ('Li', 3); testassertCount ('dirección', 0);

    }).correr( función () {
        test.done ();});});
```

assertExists ()

Firma: assertExists (selector String [, mensaje String])

Afirma que un elemento que coinciden con el proporcionado *selector de expresión* existe en el entorno de DOM remoto:

```
casper.test.begin('assertExists () pruebas', 1, función ( prueba) {
    casper.start().entonces( función () {
        esta .setContent ('<Div class = "cielo"> cerveza </ div>'); (test.assertExists '.cielo');

    }).correr( función () {
        test.done ();});});
```

Ver también:

[assertDoesntExist \(\)](#)

[assertFalsy \(\)](#)

Firma: assertFalsy (sujeto Mixed [, mensaje String])

Nuevo en la versión 1.0.

Afirma que es un tema determinado Falsy .

Ver también:

[assertTruthy \(\)](#)

[assertField \(\)](#)

Firma: assertField (String | entrada de objetos, cadena esperada [, String mensaje, Opciones de objeto])

Afirma que un campo forma fi dada tiene el valor proporcionado con el nombre de entrada o *selector de expresión* :

```
casper.test.begin('AssertField () prueba', 1, función ( prueba) {
    casper.start('Http://www.google.fr', función () {
        esta . llenar('forma [name = "GS"]', {Q : 'plaf'}, falso );
        test.assertField ('Q', 'plaf' );
    }).correr( función () {
        test.done (); });
});

// uso de Path con el tipo 'css'
casper.test.begin('AssertField () prueba', 1, función ( prueba) {
    casper.start('Http://www.google.fr', función () {
        esta . llenar('forma [name = "GS"]', {Q : 'plaf'}, falso );
        test.assertField ({type : 'Css', camino : 'q.foo'}, 'plaf' );
    }).correr( función () {
        test.done (); });
});
```

Nuevo en la versión 1.0.

Esto también funciona con cualquier tipo de entrada: seleccionar, área de texto, etc Nuevo en la versión 1.1. los *opciones* parámetro permite configurar las opciones para utilizar con [ClientUtils # GetFieldValue \(\)](#). *entrada* parámetro introspección si o no una *tipo* clave se pasa con *XPath* o *css* y una propiedad *camino* especi fi junto con él.

[assertFieldName \(\)](#)

Firma: assertFieldName (String inputName, cadena esperada [, String mensaje, Opciones de objeto])

Nuevo en la versión 1.1-beta3.

Afirma que un campo forma si dada tiene el valor proporcionado:

```
casper.test.begin ('assertFieldName () pruebas', 1 , función ( prueba) {
    casper.start ('Http://www.google.fr' , función () {
        esta . llenar( 'forma [name = "GS"]' , {Q : 'plaf' } , falso );
        test.assertFieldName ( 'Q' , 'plaf' , 'No plop' , {FormSelector : 'Plopper' } ); }).correr( función () {
            test.done (); });
});
```

assertFieldCSS ()

Firma: assertFieldCSS (String cssSelector, cadena esperada, String mensaje)

Nuevo en la versión 1.1.

Afirma que forma un campo si dada tiene el valor proporcionado dado un selector CSS:

```
casper.test.begin ('assertFieldCSS () pruebas', 1 , función ( prueba) {
    casper.start ('Http://www.google.fr' , función () {
        esta . llenar( 'forma [name = "GS"]' , {Q : 'plaf' } , falso );
        test.assertFieldCSS ( 'Q' , 'plaf' , 'No plop' , 'Input.plop' );
    }).correr( función () {
        test.done (); });
});
```

assertFieldXPath ()

Firma: assertFieldXPath (String xpathSelector, cadena esperada, Cadena
mensaje)

Nuevo en la versión 1.1.

Afirma que forma un campo si dada tiene el valor proporcionado dado un selector XPath:

```
casper.test.begin ('assertFieldXPath () pruebas', 1 , función ( prueba) {
    casper.start ('Http://www.google.fr' , función () {
        esta . llenar( 'forma [name = "GS"]' , {Q : 'plaf' } , falso );
        test.assertFieldXPath ( 'Q' , 'plaf' , 'No plop' , '/ Html / cuerpo / forma [0] /
        .. de entrada [1]' ); }).correr( función () {
            test.done (); });
});
```

assertHttpStatus ()

Firma: assertHttpStatus (estado Number [, mensaje String])

Afirma que la corriente código de estado HTTP es el mismo que el que se pasa como argumento:

```
casper.test.begin ('Casperjs.org está en marcha' , 1 , función ( prueba) {
    casper.start ('Http://casperjs.org/' , función () {
        test.assertHttpStatus ( 200 );
```

```
}).correr( función () {
    test.done ();});});
```

`assertMatch ()`

Firma: `assertMatch (sujeto mixto, patrón RegExp [, mensaje String])`

Afirma que una cadena entregada coincide con un javascript proporcionado RegExp patrón:

```
casper.test.assertMatch ( 'Chuck Norris' , / ^ Mandril / i , nombre 'Chuck Norris ' es Chuck
  -> );
```

Ver también:

- `assertUrlMatch ()`
- `assertTitleMatch ()`

`assertNot ()`

Firma: `assertNot (sujeto mixto [, mensaje String])`

Afirma que el sujeto pasó resuelve en cierta valor Falsy :

```
casper.test.assertNot ( falso , " Universo está todavía en funcionamiento" );
```

Ver también:

`afirmar()`

`assertNotEquals ()`

Firma: `assertNotEquals (TestValue mezclado, mezclado esperado [, mensaje String])`

Nuevo en la versión 0.6.7. Afirma que los dos valores son no estrictamente

igual a:

```
(casper.test.assertNotEquals cierto , " cierto" );
```

Ver también:

`assertEquals ()`

`assertNotVisible ()`

Firma: `assertNotVisible (selector String [, mensaje String])`

Afirma que el elemento de adaptación el proporcionado `selector de expresión` no es visible:

```
casper.test.begin ( 'AssertNotVisible () prueba' , 1 , función ( prueba) {
    casper.start (). entonces ( función () {
        esta . setContent ( '<Div class = estilo "foo" = "display: none> Boo </ div>' ); test.assertNotVisible ( '.foo' );
```

```
});correr( función () {
    test.done ();});});
```

Ver también:

- [assertVisible \(\)](#)
- [assertAllVisible \(\)](#)

[assertRaises \(\)](#)

Firma: assertRaises (función fn, args array [, mensaje String])

Afirma que la función prevista llamada con los parámetros dados plantea un javascript Error:

```
(casper.test.assertRaises función ( throwit) {
    Si ( throwit) {
        arrojar nueva Error ( 'Tirado' );
    }}, [ cierto ], ' Error se ha planteado ' .);

(casper.test.assertRaises función ( throwit) {
    Si ( throwit) {
        arrojar nueva Error ( 'Tirado' );
    }}, [ falso ], ' Error se ha planteado ' .); // falla
```

[assertSelectorDoesntHaveText \(\)](#)

Firma: assertSelectorDoesntHaveText (selector de cadena, cadena de texto [, string mensaje])

Afirma que el texto dado no existe en todos los elementos del juego proporcionada *selector de expresión* :

```
casper.test.begin ( 'AssertSelectorDoesntHaveText () prueba' , 1 , función ( prueba) {
    casper.start ( 'Http://google.com/' , función () {
        test.assertSelectorDoesntHaveText ( 'título' , 'Yahoo!' );
    }).correr( función () {
        test.done ();});});
```

Ver también:

[assertSelectorHasText \(\)](#)

[assertSelectorHasText \(\)](#)

Firma: assertSelectorHasText (selector de cadena, cadena de texto [, String mensaje])

Afirma que existe texto dado en los elementos que coinciden con el proporcionado *selector de expresión* :

```
casper.test.begin('AssertSelectorHasText () prueba', 1, función ( prueba) {
    casper.start('Http://google.com/', función () {
        test.assertSelectorHasText('título', 'Google');
    }).correr( función () {
        test.done();});});
```

Ver también:

[assertSelectorDoesntHaveText \(\)](#)

[assertResourceExists \(\)](#)

Firma: assertResourceExists (Función testFx [, mensaje String])

los testFx función se ejecuta contra todos los activos cargados y pasa la prueba cuando coincide con al menos un recurso:

```
casper.test.begin('assertResourceExists () pruebas', 1, función ( prueba) {
    casper.start('Http://www.google.fr', función () {
        (test.assertResourceExists función ( recurso) {
            regreso resource.url.match('Logo3w.png');
       });});}).correr( función () {
    test.done();});});
```

Corta:

```
casper.test.begin('assertResourceExists () pruebas', 1, función ( prueba) {
    casper.start('Http://www.google.fr', función () {
        (test.assertResourceExists 'Logo3w.png');
    }).correr( función () {
        test.done();});});
```

Insinuación: Consulte la documentación de [Casper.resourceExists \(\)](#).

[assertTextExists \(\)](#)

Firma: assertTextExists (String esperado [, mensaje String])

Afirmá que el cuerpo contenido de texto sin formato contiene la cadena dada:

```
casper.test.begin('assertTextExists () pruebas', 1, función ( prueba) {
    casper.start('Http://www.google.fr', función () {
        (test.assertTextExists 'Google', 'Cuerpo de la página contiene \'google\'');
    }).correr( función () {
        test.done();});});
```

Ver también:

[assertTextDoesntExist \(\)](#)

[assertTextDoesntExist \(\)](#)

Firma: assertTextDoesntExist (String inesperado [, mensaje String])

Nuevo en la versión 1.0. Afirma que el cuerpo contenido de texto sin formato no contiene la cadena

dada:

```
casper.test.begin ('assertTextDoesntExist () pruebas', 1, función ( prueba) {
    casper.start ('Http://www.google.fr', función () {
        test.assertTextDoesntExist ('Bing', 'Cuerpo de la página no contiene \'Bing\' ');
    }).correr( función () {
        test.done (); });
});
```

Ver también:

[assertTextExists \(\)](#)

[assertTitle \(\)](#)

Firma: assertTitle (String esperado [, mensaje String])

Afirma que el título de la página a distancia es igual a la esperada:

```
casper.test.begin ('AssertTitle () prueba', 1, función ( prueba) {
    casper.start ('Http://www.google.fr', función () {
        test.assertTitle ('Google', 'Google.fr tiene el título correcto' );
    }).correr( función () {
        test.done (); });
});
```

Ver también:

[assertTitleMatch \(\)](#)

[assertTitleMatch \(\)](#)

Firma: assertTitleMatch (patrón RegExp [, mensaje String])

Afirma que el título de la página remoto coincide con el patrón de expresión regular, siempre que:

```
casper.test.begin ('assertTitleMatch () pruebas', 1, función ( prueba) {
    casper.start ('Http://www.google.fr', función () {
        test.assertTitleMatch ( / Google / , 'Google.fr tiene un título bastante predecible' );
    }).correr( función () {
        test.done (); });
});
```

Ver también:

[assertTitle \(\)](#)

assertTruthy ()

Firma: assertTruthy (sujeto Mixed [, mensaje String])

Nuevo en la versión 1.0.

Afirma que es un tema determinado Truthy .

Ver también:

[assertFalsy \(\)](#)

assertType ()

Firma: assertType (entrada mixta, tipo String [, mensaje String])

Afirma que la entrada proporcionada es del tipo dado:

```
casper.test.begin ('assertType () pruebas', 1 , función suite (prueba) {
    test.assertType ( 42 , "número" , "Okay, 42 es un número" ); test.assertType ([ 1 , 2 , 3 ], "formación" , "Podemos probar para
    matrices también!" ); test.done ();});
```

Nota: nombres de los tipos se expresan siempre en minúsculas.

assertInstanceOf ()

Firma: assertInstanceOf (entrada mixta, la función de constructor [, mensaje String])

Nuevo en la versión 1.1.

Afirma que la entrada proporcionada es del constructor dado:

```
función Vaca() {
    esta . mugir = función moo () {
        regreso ' ¡mugir! ' ;});}

casper.test.begin ('AssertInstanceOf () prueba', 2 , función suite (prueba) {
    var margarita = nuevo Vaca();
    test.assertInstanceOf (margarita, de la vaca, "Ok, margarita es una vaca." ); test.assertInstanceOf ([ "mugir" , "abucheo" ], Formación , "Podemos
    probar para matrices también!" ); test.done ();});
```

assertUrlMatch ()

Firma: assertUrlMatch (patrón Regexp [, mensaje String])

Afirma que la URL de la página actual coincide con el patrón de expresión regular, siempre que:

```
casper.test.begin ('assertUrlMatch () pruebas', 1 , función ( prueba) {
    casper.start ( 'Http://www.google.fr' , función () {
        test.assertUrlMatch ( / ^ Http: \ / \ // , 'Google.fr se sirve en http: /' );
```

```
});correr( función () {
    test.done ();});});
```

assertVisible ()

Firma: assertVisible (selector String [, mensaje String])

Afirmá que al menos un elemento que coinciden con el proporcionado *selector de expresión* es visible:

```
casper.test.begin ('AssertVisible () prueba', 1 , función ( prueba) {
    casper.start ('Http://www.google.fr' , función () {
        test.assertVisible ('H1');
    }).correr( función () {
        test.done ();});});
```

Ver también:

- [assertAllVisible \(\)](#)
- [assertNotVisible \(\)](#)

assertAllVisible ()

Firma: assertAllVisible (selector String [, mensaje String])

Afirmá que todos los elementos que coinciden con los proporcionados *selector de expresión* son visibles:

```
casper.test.begin ('AssertAllVisible () prueba', 1 , función ( prueba) {
    casper.start ('Http://www.google.fr' , función () {
        test.assertAllVisible ('De entrada [type = "submit"]');
    }).correr( función () {
        test.done ();});});
```

Ver también:

- [assertVisible \(\)](#)
- [assertNotVisible \(\)](#)

empezar()

firmas:

- begin (Descripción de cuerdas, número previsto, salón de actos)
- begin (Descripción de cuerdas, salón de actos)
- begin (Descripción de cuerdas, número previsto, de configuración de objetos)
- begin (Descripción de cuerdas, objetos de configuración)

Nuevo en la versión 1.1. Inicia un conjunto de < planeado> pruebas (si se ha definido). los suite devolución de llamada obtendrá la corriente Ensayador instancia como su primer argumento:

```

función Vaca() {
    esta . segado = falso ;
    esta . mugir = función moo () {
        esta . segado = cierto ; // Estado mootable: no hagas eso
        regreso " ¡mugir! ";
    }

    // unidad de estilo de caso de prueba sincrónica
casper.test.begin ('Vaca puede MOO', 2 , función suite (prueba) {
    var vaca = nuevo Vaca(); test.assertEquals (cow.moo (), ' ¡mugir! ');
    test.assert (cow.mowed); test.done ();
}

```

Nota: los planificado argumento es especialmente útil en el caso de una secuencia de comandos de prueba dado se interrumpe bruscamente la deja con ninguna manera obvia de conocerla y el estado erróneamente éxito.

Un ejemplo más asíncrono:

```

casper.test.begin ('Casperjs.org es navegable', 2 , función suite (prueba) {
    casper.start ('Http://casperjs.org' , función () {
        (test.assertTitleMatches / casperjs / i );
        esta . clickLabel ('Pruebas' );
    });

    casper.then ( función () {
        (test.assertUrlMatches /testing\.html$/ );
    });

    casper.run ( función () {
        test.done (); });
}

```

Importante: [hecho\(\)](#) debe ser llamado para poner fin a la suite. Esto es especialmente importante cuando se hace pruebas de manera asíncrona a asegurar que se llama cuando todo en realidad se ha realizado.

Ver también:

[hecho\(\)](#)

Tester # begin () También acepta un objeto con configuración de prueba, lo que puede añadir preparar() y demoler() métodos:

```

// vaca-Test.js
casper.test.begin ('Vaca puede MOO', 2 , { preparar : función ( prueba ) {

    esta . vaca = nuevo Vaca();
}

```

```
demoler : función ( prueba) {
    esta . cow.destroy (); };

prueba : función ( prueba) {
    (test.assertEquals esta . cow.moo (), '¡mugir!'); test.assert ( esta . cow.mowed);
    test.done (); });


```

colorear ()

Firma: colorear (String mensaje, el estilo de cadena)

Render una salida coloreada. Básicamente un método proxy para Casper.Colorizer # colorean () .

comentario()

Firma: comentario (String mensaje)

Escribe un mensaje con formato de estilo comentario a la salida estándar:

```
casper.test.comment ( "Hola, soy un comentario" );
```

hecho()

Firma: hecho()

Cambiado en la versión 1.1: planificado parámetro es obsoleto bandera de un conjunto de

pruebas se inició con *empezar()* como procesado:

```
casper.test.begin ( 'Mi banco de pruebas' , 2 , función ( prueba) {
    test.assert ( cierto );
    test.assertNot ( falso );
    test.done (); });


```

Más de forma asíncrona:

```
casper.test.begin ( 'Casperjs.org es navegable' , 2 , función suite (prueba) {
    casper.start ( 'Http://casperjs.org' , función () {
        (test.assertTitleMatches / casperjs / i );
        esta . clickLabel ( 'Pruebas' );
    });

    casper.then ( función () {
        (test.assertUrlMatches /testing\.html$/ );
    });

    casper.run ( función () {
        test.done (); });
});
```

Ver también:

[empezar\(\)](#)

error()

Firma: error (mensaje String)

Escribe un mensaje con formato de estilo de error a la salida estándar:

```
casper.test.error ( "Hola, soy un error" );
```

fallar()

Firma: fallar (String mensaje [, opción Objeto])

Agrega una entrada de prueba fallida a la pila:

```
casper.test.fail ( "Georges W. Bush" ); casper.test.fail ( "Aquí va un mensaje muy largo y expresivo" , {nombre : "shortfacts" } )
```

...);

Ver también:

[pasar\(\)](#)

FormatMessage ()

Firma: FORMATMESSAGE (String mensaje, el estilo de cadena)

Formatea un mensaje para resaltar algunas partes del mismo. Sólo se utiliza internamente por el probador.

getFailures ()

Firma: getFailures ()

Nuevo en la versión 1.0.

En desuso desde la versión 1.1. Recupera los fallos de

banco de pruebas actual:

```
(casper.test.assertEquals cierto , falso );
exigir( 'utils' ) .dump (casper.test.getFailures()); casper.test.done();
```

Eso le dará algo como esto:

```
$ Casperjs archivo de prueba prueba prueba-getFailures.js:
Prueba-getFailures.js FALLO Sujeto es igual al valor esperado

#      Tipo: assertEquals
#      Asunto: verdadera
#      espera: {falsa

    "Longitud": 1,
```

```
"casos": [
  {
    "Éxito": false, "type": "assertEquals",
    "Estándar": "A reserva es igual al valor esperado", "archivo": "prueba de getFailures.js",
    "valores": {
      "Objeto": true, "espera": false}}]}]
```

FALLO 1 pruebas ejecutadas, 0 Pasadas, 1 fallidos.

Los detalles de la prueba fallida 1:

En c.js: 0
assertEquals: Sujeto es igual al valor esperado

Nota: En CasperJS 1.1, puede registrado fallos de las pruebas al escuchar el probador fallar evento:

```
var fracasos = [];

casper.test.on( "fallar" , función ( fracaso ) {
  failures.push ( fallo );});
```

getPasses ()

Firma: **getPasses ()**

Nuevo en la versión 1.0.

En desuso desde la versión 1.1.

Recupera un informe de casos de prueba con éxito en el banco de pruebas actual:

```
(casper.test.assertEquals cierto , cierto );
exigir( 'utils' ).dump (casper.test.getPasses()); casper.test.done();
```

Eso le dará algo como esto:

```
$ Test test casperjs - Archivo de prueba getPasses.js : prueba - getPasses.js
PASS Sujeto es igual al valor esperado {

  "longitud" : 1 ,
  "casos" : [
    {
      "éxito" : cierto ,
      "tipo" : "assertEquals" ,
      "estándar" : "Sujeto es igual al valor esperado" ,
      "archivo" : "test-getPasses.js" ,
```

```

    "valores" : {
        "tema" : cierto ,
        "esperado" : cierto
    } }]}) PASAR 1 pruebas ejecutadas, 1 pasado, 0 ha
fallado.

```

Nota: En CasperJS 1.1, puede registrado éxitos de prueba al escuchar el probador éxito evento:

```

var éxitos = [];

casper.test.on ( "éxito" , función ( éxito ) {
    successes.push ( éxito );
}

```

info ()

Firma: info (String mensaje)

Escribe un mensaje con formato de información de estilo a la salida estándar:

```
casper.test.info ( "Hola, soy un mensaje informativo." );
```

pasar()

Firma: pase (mensaje String)

Agrega una entrada de prueba con éxito a la pila:

```
casper.test.pass ( "Barack Obama" );
```

Ver también:

fallar()

renderResults ()

Firma: renderResults (salida booleana, estado Number, String guardar)

Render resultados de las pruebas, guardar los resultados en un formato XUnit si l, y opcionalmente sale PhantomJS:

```
casper.test.renderResults ( cierto , 0 , 'Test-results.xml' );
```

Nota: Este método no se va a llamar cuando se utiliza el prueba casperjs de comandos (consulte la documentación de [pruebas](#)), donde se hace automáticamente.

preparar()

Firma: SETUP ([Función fn])

De fi ne una función que se ejecutará antes de cada prueba de fi nido usando *empezar()* :

```
casper.test.setUp ( función () {
    casper.start (). userAgent ( 'Mosaic 0,1' );
});
```

Para llevar a cabo las operaciones asíncronas, utilice el hecho argumento:

```
casper.test.setUp ( función ( hecho ) {
    casper.start ( 'Http://foo' ). entonces( función () {
        // ...
    }) Run ( hecho );});
```

Advertencia: No especifique el hecho argumento si usted no tiene intención de utilizar el método de forma asíncrona.

Ver también:

demoler()

omitir()

Firma: saltar (Número nb, mensaje String)

Omite un número dado de pruebas previstas:

```
casper.test.begin ( 'Skip pruebas' , 4 , función ( prueba ) {
    test.assert ( cierto , ' Primera prueba ejecutado' ); test.assert ( cierto , ' Segunda prueba ejecutado' );
    test.skip ( 2 , 'Dos pruebas omiten' ); test.done ();
});
```

demoler()

Firma: tearDown ([Función fn])

De fi ne una función que se ejecutará después de cada prueba de fi nido usando *empezar()* :

```
casper.test.tearDown ( función () {
    casper.echo ( 'Nos vemos' );
});
```

Para llevar a cabo las operaciones asíncronas, utilice el hecho argumento:

```
casper.test.tearDown ( función ( hecho ) {
    casper.start ( 'Http://hola/adiós' ). entonces( función () {
        // ...
    }) Run ( hecho );});
```

Advertencia: No especifique el hecho argumento si usted no tiene intención de utilizar el método de forma asíncrona.

Ver también:

[preparar\(\)](#)

los utils módulo

Este módulo proporciona funciones de ayuda simples, algunos de ellos muy específico a CasperJS sin embargo.

referencia de funciones

El uso es bastante sencillo:

```
var utils = exigir('utils');

utils.dump ({plop : 42});
```

betterTypeOf ()

Firma: betterTypeOf (entrada)

Proporciona una mejor tipo de equivalente operador, por ejemplo. capaz de recuperar la Formación tipo.

betterInstanceOf ()

Nuevo en la versión 1.1.

Firma: betterInstanceOf (entrada, constructor)

Proporciona una mejor en vez de equivalente del operador, es capaz de recuperar la Formación instancia o para hacer frente a la herencia.

tugurio()

Firma: dump (valor)

Vuelca una JSON representación del argumento pasado a la salida estándar. Útil para *la depuración de secuencias de comandos*.

FileExt ()

Firma: FileExt (archivo)

Recupera la extensión de nombre de archivo pasado fi.

fillBlanks ()

Firma: fillBlanks (texto, pad)

Rellena una cadena con espacios finales para que coincida almohadilla longitud.

formato()

Firma: formato (f)

Da formato a una cadena contra argumentos pasados. sprintf equivalente.

Nota: Se trata de un puerto de nodejs util.format ()�

getPropertyPath ()

Firma: getPropertyPath (Object obj, trayectoria String)

Nuevo en la versión 1.0.

Recupera el valor de una propiedad extranjera de objetos utilizando una cadena de ruta separada por puntos:

```
var cuenta = {
    nombre de usuario : 'arrojar' ,
    habilidades : {
        patada : {
            casa de máquinas : cierto
        }
    }
}

utils.getPropertyPath (Cuenta, 'Skills.kick.roundhouse' ); // cierto
```

Advertencia: Esta función no maneja nombres clave de objeto que contienen un punto.

hereda ()

Firma: hereda (ctor, superCtor)

Hace un constructor que hereda de otra. Útil para la subclasificación y extensión .

Nota: Se trata de un puerto de nodejs util.inherits ()�

isArray ()

Firma: isArray (valor)

Verifica si el argumento pasado es una instancia de Formación.

isCasperObject ()

Firma: isCasperObject (valor)

Verifica si el argumento pasado es una instancia de Casper.

`isClipRect ()`

Firma: `isClipRect (valor)`

Comprueba si es un argumento pasado clipRect objeto.

`isFalsy ()`

Firma: `isFalsy (sujeto)`

Nuevo en la versión 1.0. Las devoluciones

sujetas falsiness .

`isFunction ()`

Firma: `isFunction (valor)`

Verifica si el argumento pasado es una función.

`isJsFile ()`

Firma: `isJsFile (archivo)`

Comprueba si transcurrido nombre de archivo fi es una Javascript (comprobando si tiene una. js o. café fi le extensión).

`es nulo()`

Firma: `isNull (valor)`

Comprueba si es un argumento pasado nulo.

`es número()`

Firma: `ISNUMBER (valor)`

Verifica si el argumento pasado es una instancia de Número.

`isObject ()`

Firma: `isObject (valor)`

Verifica si el argumento pasado es un objeto.

`isString ()`

Firma: `isString (valor)`

Verifica si el argumento pasado es una instancia de Cuerda.

`isTruthy ()`

Firma: `isTruthy (sujeto)`

Nuevo en la versión 1.0. Las devoluciones

sujetas `truthiness`.

`isType ()`

Firma: `isType (lo que, tipo)`

Verifica si el argumento pasado tiene su tipo que coincide con el tipo argumento.

`es indefinido()`

Firma: `isUndefined (valor)`

Verifica si el argumento es pasado indefinido.

`isWebPage ()`

Firma: `isWebPage (lo)`

Verifica si el argumento pasado es una instancia de PhantomJS nativos' Página web objeto.

`mergeObjects ()`

Firma: `mergeObjects (origen, añaden [, objeto opta])`

Combinar dos objetos de forma recursiva. Añadir `opts.keepReferences` si no se necesita la clonación de los objetos internos.

`nodo()`

Firma: `nodo (nombre, atributos)`

Crea un (HT | X) elemento ML, teniendo opcional atributos adicional.

`publicar por fascículos()`

Firma: `serializar (valor)`

Serializa un valor utilizando `JSON` formato. Serializará funciones como cadenas. Útil para `depuración` y la comparación de objetos.

`único()`

Firma: `único (array)`

Recupera valores únicos dentro de una determinada Formación.

CAPÍTULO 7

Escribiendo módulos CasperJS

A partir de 1.1, se basa en CasperJS nativa PhantomJS' exigir() función internamente a pesar de que tuvo que ser parcheado con el fin de permitir que requiere módulos Casper utilizando su nombre completo, por ejemplo. require ('Casper').

Así que si va a escribir sus propios módulos y utilizar los casperjs' de ellos, asegúrese de llamar a la patchRequire () función:

```
// mi módulo, almacenada en universe.js // parches PhantomJS'
require ()

var exigir = patchRequire (requiere);

// ahora está listo para ir
var utils = exigir( 'utils' );
var magia = 42 ; exports.answer = función () {

    regreso utils.format ( "Es% d" , magia);
};
```

Advertencia: Al utilizar CoffeeScript global.require debe pasar a patchRequire () en lugar de sólo requiere: require = patchRequire global.require

```
utils = exigir 'utils'
magia = 42
exports.answer = ->
    utils.format "Es $ {} mágica"
```

A partir de su escritura de Casper raíz:

```
var universo = exigir( './universo' ); console.log (universe.answer()); // imprime "Es 42"
```

Nuevo en la versión 1.1 ..

Insinuación: CasperJS permite el uso de módulos nodejs instalados a través de [NPM](#). Tenga en cuenta que, dado que utiliza CasperJS su propio entorno de JavaScript, los módulos de la NGP que utilizan características ganglios específicos no funcionarán bajo CasperJS.

A modo de ejemplo, vamos a instalar el [guion bajo](#) biblioteca:

```
$ NPM instalar subrayado
```

Entonces, exigir como lo haría con cualquier módulo compatible con otros nodejs:

```
//npm-underscore-test.js
var _ = exigir('guion bajo');
var Casper = exigir('Casper').crear();
var URLs = _.uniq([
    'Http://google.com/',
    'Http://docs.casperjs.org/',
    'Http://google.com/'
]);

casper.start().eachThen(URL, función (respuesta) {
    esta .thenOpen(response.data, función (respuesta) {
        esta .eco(esta .getTitle()); });
});

casper.run();
```

Por último, es probable que obtener algo como esto:

```
$ casperjs NPM-subrayado-Test.js Google
documentación CasperJS | CasperJS 1.1.0-DEV documentación
```

CAPÍTULO 8

Eventos y Filtros

CasperJS proporciona una *controlador de eventos* muy similar a la `nodejs' uno`; En realidad se toma prestado la mayor parte de su código base. También añade CasperJS *filtros*, que son básicamente formas de alterar los valores de forma asíncrona.

El uso de eventos es bastante sencillo si eres un desarrollador de nodo, o si ha trabajado con cualquier sistema evented antes:

```
var Casper = exigir('Casper').crear();

casper.on('Resource.received', función (recurso) {
    casper.echo(resource.url);
});
```

Que emite es el propietario eventos

Por supuesto se puede emitir sus propios eventos, utilizando la `Casper.emit()` método:

```
var Casper = exigir('Casper').crear();

// escuchar un evento personalizado
casper.on('Google.loaded', función () {
    esta . eco('Título de la página es Google' + esta . getTitle());
});

casper.start('Http://google.com', función () {
    // emisión de un evento personalizado
    esta . emitir('Google.loaded');
});

casper.run();
```

Extracción de eventos

También puede eliminar eventos. Esto es particularmente útil cuando se ejecuta una gran cantidad de pruebas en las que pueda necesitar para añadir y eliminar eventos diferentes para diferentes pruebas:

```
var Casper = exigir( 'Casper' ).crear();

// función de escucha para los recursos solicitados
var oyente = función ( recurso, petición ) {
    esta . echo (resource.url); }

// escuchar a todas las solicitudes de recursos
casper.on ( "Resource.requested" , Oyente);

// cargar la página principal de Google
casper.start ( 'Http://google.com/' , función () {
    esta . eco( esta . getTitle () );
});

casper.run (). entonces ( función () {
    // eliminar el detector de eventos
    esta . removeListener ( "Resource.requeste
});
```

Aquí está un ejemplo de cómo utilizar esto en una prueba casperjs dentro de la función tearDown:

```
var currentRequest;

// oyente de recursos
función onResourceRequested (requestData, solicitud) {
    Si (/^jQuery \.min \.js /.test (requestData.url)) {currentRequest = requestData; }

casper.test.begin ('Prueba JQuery', 1, { preparar : función () {

    // Una el oyente de recursos
    casper.on ('Resource.requested', OnResourceRequested);
    },

demoler : función () {
    // eliminar el oyente de recursos
    casper.removeListener ('Resource.requested', OnResourceRequested); currentRequest = indefinido;

    },

prueba : función ( prueba) {
    casper.start ('Http://casperjs.org', función () {
        test.assert (currentRequest !== indefinido, " JQuery existe ");
    });

    casper.run ( función () {
        test.done (); });
});
```

Eventos referencia

espalda

argumentos: Ninguna

Emitido cuando se le pide al navegador integrado que retroceder un paso en su historia.

capture.saved

argumentos: archivo de destino

Emitido cuando una imagen de pantalla se ha capturado.

hacer clic

argumentos: selector

Emitida cuando el Casper.click () método ha sido llamado.

complete.error

argumentos: error

Nuevo en la versión 1.1.

Emitido cuando una devolución de llamada completa con error ha.

Por defecto, CasperJS no escucha a este evento, que tiene que declarar sus propios oyentes con la mano:

```
casper.on ('Complete.error' , función ( err ) {  
    esta . morir( "Devolución de llamada completo ha fallado:" + err);});
```

morir

argumentos: mensaje, el estado

Emitida cuando el Casper.die () método ha sido llamado.

downloaded.file

argumentos: targetPath

Emitida cuando un expediente ha sido descargado por [Casper.download \(\)](#); objetivo contendrá la ruta de acceso al fi l descargado.

downloaded.error

argumentos: url

Emitida cuando un expediente ha encoutered un error cuando se descargan por [Casper.download \(\)](#); url contendrá la URL de la descargado fi l.

error

argumentos: msg, traza inversa

Nuevo en la versión 0.6.9.

Emitida cuando un error no ha sido capturado de forma explícita dentro del entorno CasperJS / PhantomJS. Hacer básicamente lo PhantomJS' onError () controlador nativo hace.

salida

argumentos: estado

Emitida cuando el Casper.exit () método ha sido llamado.

fileDownloadError

argumentos: error

Emite cuando se produce un error en la descarga de archivo.

fileToDownload

argumentos: Objeto

Emite cuando se produce la respuesta con una cabecera Content-Disposition.

llenar

argumentos: selector, Vals, se someten

Emite cuando un formulario se llena usando el Casper.fill () método.

adelante

argumentos: Ninguna

Emite cuando se le pide al navegador integrado para avanzar un paso más en su historia.

frame.changed

argumentos: nombre, estado

Emitida cuando se cambia el cuadro actual con Casper.withPopup, Casper.switchToFrame ()

http.auth

argumentos: usuario Contraseña

Emitido cuando se establecen parámetros de autenticación HTTP.

http.status. [code]

argumentos: recurso

Emite cuando cualquier reponse HTTP dado es recibido con el código de estado específico ed por [código], p.ej.:

```
casper.on( 'Http.status.404' , función ( recurso ) {
    casper.echo (resource.url + 'Es 404' );
})
```

load.started

argumentos: Ninguna

Emite cuando PhantomJS' WebPage.onLoadStarted devolución de llamada de evento se llama.

carga fallida

argumentos: Objeto

Emite cuando PhantomJS' WebPage.onLoadFinished callback del evento ha sido llamado y han fracasado.

load.finished

argumentos: estado

Emite cuando PhantomJS' WebPage.onLoadFinished devolución de llamada de evento se llama.

Iniciar sesión

argumentos: entrada

Emitida cuando el Casper.log () método ha sido llamado. los entrada parámetro es un objeto de esta manera:

```
{  
    nivel : "depurar",  
    espacio : "fantasma",  
    mensaje : "Un mensaje", fecha :  
        "Una instancia Fecha javascript"  
}
```

click del raton

argumentos: args

Emite cuando el ratón hacia la izquierda clic en algo o en alguna parte.

ratón hacia abajo

argumentos: args

Emitida cuando el ratón presiona en algo o en algún lugar con el botón izquierdo.

mouse.move

argumentos: args

Emite cuando el ratón se mueve hacia algo o en alguna parte.

mouse.up

argumentos: args

Emitida cuando el ratón libera el botón izquierdo sobre algo o en alguna parte.

navigation.requested

argumentos: url, navigationType, navigationLocked, isMainFrame

Nuevo en la versión 1.0.

Emitida cada vez que se ha solicitado una operación de navegación. tipos de navegación disponibles son: LinkClicked, FormSubmitted, BackOrForward, Recargar, FormResubmitted y Otro.

abierto

configuración de ubicación

Emitido cuando una solicitud HTTP se envía. Primer argumento de devolución de llamada es la ubicación, segundo es un objeto de configuración de la solicitud de la forma:

```
{  
    método : "enviar", los datos :  
        "Foo = 42 y mandril = Norris"  
}
```

page.created

argumentos: página

Emite cuando PhantomJS' Página web objeto utilizado por CasperJS ha sido creado.

page.error

argumentos: mensaje, vestigios

Emitida página cuando recuperado deja sin capturar un error de Javascript:

```
casper.on( "Page.error" , función ( msg, vestigios) {  
    esta . eco( "Error:" + msg, "ERROR" );  
});
```

page.initialized

argumentos: Página web

Emite cuando PhantomJS' Página web objeto utilizado por CasperJS se ha inicializado.

page.resource.received

argumentos: ResponseData

Emitido cuando se ha recibido la respuesta HTTP correspondiente a la URL requerida actual:

```
casper.on ('Page.resource.received', función (responseData) {  
    esta . echo (responseData.url);});
```

Propiedades de ResponseData son:

- **camé de identidad:** el número del recurso solicitado
 - **url:** la url del recurso
 - **hora:** un objeto Date
 - **encabezados:** la lista de cabeceras (lista de objetos {nombre: "", valor: ""})
 - **tamaño corporal:** el tamaño del contenido recibido (puede aumentar durante múltiple llamada de la devolución de llamada)
 - **tipo de contenido:** el tipo de contenido del recurso
 - **contentCharset:** el juego de caracteres usado por el contenido de los recursos (slimerjs solamente).
 - **Redireccionar URL:** si la solicitud ha sido redirigido, esta es la URL redireccionada
 - **escenario:** "Inicio", "fin" o "" para la porción intermedia de datos
 - **estado:** el código de respuesta HTTP (200 ..)
 - **statusText:** el texto de respuesta HTTP para el estado ("Ok" ...)
 - **de referencia:** El árbitro url (slimerjs solamente)
 - **cuerpo:** el contenido, puede cambiar durante la llamada múltiple para la misma petición (slimerjs solamente).
 - **httpVersion.major:** la mayor parte de la versión del protocolo HTTP (slimerjs solamente).
 - **httpVersion.minor:** la parte menor de la versión del protocolo HTTP (slimerjs solamente).
-

page.resource.requested

argumentos: solicitud

Emite cuando una nueva solicitud HTTP se realiza para abrir la URL requerida. Nuevo en la versión

1.1.

argumentos: RequestData, networkRequest

También puede abortar peticiones de:

```
casper.on( 'Page.resource.requested', función ( requestData, networkRequest ) {
    Si ( requestData.url.indexOf( 'Http://adserver.com' ) === 0 ) {NetworkRequest.abort (); };
```

Propiedades de RequestData son:

- **id:** carné de identidad: el número del recurso solicitado
- **método:** el método HTTP ("get", "post" ..)
- **url:** la url del recurso
- **hora:** un objeto Date
- **encabezados:** la lista de cabeceras (lista de objetos {nombre: "", valor: ""})
- **Postdata:** una cadena que contiene el cuerpo de la solicitud, cuando el método es "post" o "put" (SlimerJS 0.9) El objeto networkRequest

tiene dos métodos:

- **abortar():** llamar para cancelar la solicitud. onResourceReceived y onLoadFinished serán llamados.
- **changeUrl (URL):** abortar la solicitud actual y hacer una redirección de inmediato a la URL dada.
- **setHeader (clave, valor, fusión):** le permite establecer una cabecera en la petición HTTP. Si el valor es nulo o una serie vacía, se retira la cabecera. El parámetro de mezcla (sólo disponible en SlimerJS), es un valor booleano: true para combinar el valor dado con un valor existente para esta cabecera. Si es falso, el valor antiguo es reemplazado por el nuevo. (Introducción: SlimerJS 0.9)

popup.created

argumentos: Página web

Emitido cuando se ha abierto una nueva ventana.

popup.loaded

argumentos: Página web

Emitido cuando se ha cargado una nueva ventana.

popup.closed

argumentos: Página web

Emitido cuando una nueva ventana abierta se ha cerrado.

remote.alert

argumentos: mensaje

Emitted when a remote alert() call has been performed.

remote.callback

Arguments: data

Emitted when a remote `window.callPhantom(data)` call has been performed.

remote.longRunningScript

Arguments: WebPage

Emitted when any remote longRunningScript call has been performed. You have to call stopJavaScript

method

```
casper.on('remote.longRunningScript', function stopLongScript(webpage) {
    webpage.stopJavaScript();
    return true;
});
```

remote.message

Arguments: msg

Emitted when any remote console logging call has been performed.

resource.error

Arguments: resourceError

Emitted when any requested resource fails to load properly. The received resourceError object has the following properties:

- errorCode: error code
 - errorString: error description
 - url: resource url
 - id: resource id
-

resource.received

Arguments: resource

Emitted when any resource has been received.

Arguments: responseData

Emitted when the HTTP response corresponding to current required url has been received:

```
casper.on('resource.received', function (responseData) {
    this.echo(responseData.url);});
```

Properties of responseData are:

- id: the number of the requested resource
 - url: the url of the resource
 - time: a Date object
 - headers: the list of headers (list of objects {name:"", value:""})
 - bodySize: the size of the received content (may increase during multiple call of the callback)
 - contentType: the content type of the resource
 - contentCharset: the charset used for the content of the resource (slimerjs only).
 - redirectURL: if the request has been redirected, this is the redirected url
 - stage: "start", "end" or "" for intermediate chunk of data
 - status: the HTTP response code (200..)
 - statusText: the HTTP response text for the status ("Ok" ...)
 - referrer: the referer url (slimerjs only)
 - body: the content, it may change during multiple call for the same request (slimerjs only).
 - httpVersion.major: the major part of the HTTP protocol version (slimerjs only).
 - httpVersion.minor: the minor part of the HTTP protocol version (slimerjs only).
-

resource.requested

Arguments: requestData, networkRequest

You can also abort or change requests and also update Headers

```
casper.on('resource.requested', function (requestData, networkRequest) {
    if (requestData.url.indexOf('http://adserver.com') === 0) { networkRequest.abort(); }});
```

Properties of requestData are:

- id: the number of the requested resource
- method: the http method ("get", "post" ..)

- url: the url of the resource
- time: a Date object
- headers: the list of headers (list of objects {name:"", value:""})
- postData: a string containing the body of the request, when method is "post" or "put" (SlimerJS 0.9) The networkRequest object

has two methods:

- abort(): call it to cancel the request. onResourceReceived and onLoadFinished will be called.
 - changeUrl(url): abort the current request and do an immediate redirection to the given url.
 - setHeader(key, value, merge): allows you to set an header on the HTTP request. If value is null or an empty string, the header will be removed. The merge parameter (only available on SlimerJS), is a boolean: true to merge the given value with an existing value for this header. If false, the old value is replaced by the new one. (Introduced: SlimerJS 0.9)
-

resource.timeout

Arguments: request

Emitted when the execution time of any resource has exceeded the value of settings.resourceTimeout. you can configure timeout with settings.resourceTimeout parameter. Properties of responseData are:

- id: the number of the requested resource
 - url: the url of the resource
 - errorCode: an error code: 408
 - errorMessage: the error message.
 - time: a Date object
 - headers: the list of headers (list of objects {name:"", value:""})
 - method: the http method ("get", "post"..)
-

run.complete

Arguments: None

Emitted when the whole series of steps in the stack have been executed.

run.start

Arguments: None

Emitted when Casper.run() is called.

starting

Arguments: None

Emitted when Casper.start() is called.

started

Arguments: None

Emitted when Casper has been started using Casper.start().

step.added

Arguments: step

Emitted when a new navigation step has been added to the stack.

step.bypassed

Arguments: step, step

Emitted when a new navigation step has been reached by bypass (destination, origin).

step.complete

Arguments: stepResult

Emitted when a navigation step has been executed.

step.created

Arguments: fn

Emitted when a new navigation step has been created.

step.error

Arguments: error

New in version 1.1.

Emitted when a step function has errored.

By default, CasperJS doesn't listen to this event, you have to declare your own listeners by hand:

```
casper.on('step.error', function (err) {
  this . die("Step has failed: " + err);});
```

step.start

Arguments: step

Emitted when a navigation step has been started.

step.timeout

Arguments: [step, timeout]

Emitted when a navigation step has timed out.

timeout

Arguments: None

Emitted when the execution time of the script has reached the Casper.options.timeout value.

url.changed

Arguments: url

New in version 1.0.

Emitted each time the current page url changes.

viewport.changed

Arguments: [width, height]

Emitted when the viewport has been changed.

wait.done

Arguments: None

Emitted when a Casper.wait() *operation ends*.

wait.start

Arguments: None

Emitted when a Casper.wait() operation starts.

waitFor.timeout

Arguments: [timeout, details]

Emitted when the execution time of a Casper.wait*() operation has exceeded the value of timeout. details is a property bag describing what was being waited on. For example, if waitForSelector timed out, details will have a selector string property that was the selector that did not show up in time.

Filters Reference

Filters allow you to alter some values asynchronously. Sounds obscure? Let's take a simple example and imagine you would like to alter every single url opened by CasperJS to append a foo=42 query string parameter:

```
var casper = require('casper').create();

casper.setFilter('open.location', function (location) {
    return A?+/.test(location) ? location += "&foo=42" : location += "?foo=42" ;});
```

There you have it, every single requested url will have this appended. Let me bet you'll find far more interesting use cases than my silly one ;)

Every filter methods called emit an identical event. For instance, "page.confirm" filter sends "page.confirm" event. Here's the list of all available filters with their expected return value: Filters reference

capture.target_filename

Arguments: args

Return type: String

Allows to alter the value of the filename where a screen capture should be stored.

echo.message

Arguments: message

Return type: String

Allows to alter every message written onto stdout.

fileDownload

Arguments: url, Object [filename,size,contentType]

Return type: String

Allows to alter the path for the file that must be downloaded.

log.message

Arguments: message

Return type: String

Allows to alter every log message.

open.location

Arguments: args

Return type: String

Allows to alter every url before it being opened.

page.confirm

Arguments: message

Return type: Boolean

New in version 1.0.

Allows to react on a javascript confirm() call:

```
casper.setFilter( "page.confirm" , function ( msg ) {
    return msg === "Do you like vbscript?" ? false : true ;
});
```

page.filePicker

Arguments: oldFile

Return type: String

New in version 1.4.

Allows to react on a webpage.onFilePicker call:

```
casper.setFilter( "page.filePicker" , function ( oldFile ) {
    if ( system.os.name === 'windows' ) {
        return ' C:\\Windows\\System32\\drivers\\etc\\hosts' ;
    }
    return '/etc/hosts' ;
});
```

page.prompt

Arguments: message, value

Return type: String

New in version 1.0.

Allows to react on a javascript prompt() call:

```
casper.setFilter( "page.prompt" , function ( msg, value ) {
    if ( msg === "What's your name?" ) {
        return " Chuck" ;
    }
});
```


CHAPTER 9

Logging

CasperJS allows logging using the `casper.log()` method and these standard event levels:

- debug
- info
- warning
- error

Sample use:

```
var casper = require('casper').create(); casper.log('plop', 'debug');
casper.log('plip', 'warning');
```

Now, there are two things to distinguish: log *storage* and log *display*; by default CasperJS won't print the logs to the standard output. In order to do so, you must enable the verbose Casper option:

```
var casper = require('casper').create({ verbose : true
});
```

Also, by default Casper is configured to filter logging which is under the error level; you can override this setting by configuring the `logLevel` option:

```
var casper = require('casper').create({ verbose : true ,
logLevel : 'debug'
});
```

You can also dump a JSON log of your Casper suite just by rendering the contents of the `Casper.result.log` property:

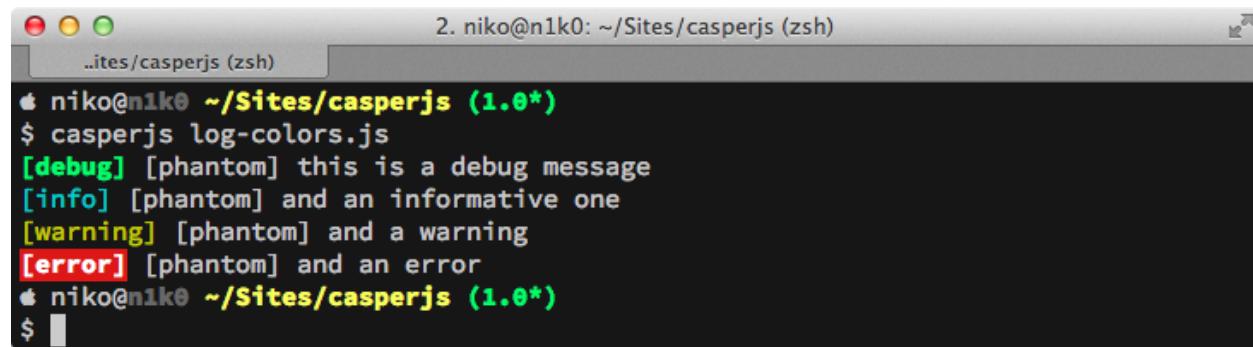
```
var casper = require('casper').create({
// ...
```

```
casper.run( function () {
    require('utils').dump( this . result.log);
    this . exit();});
```

Last, if you print log messages to the standard output using the verbose option, you'll get some fancy colors:

```
var casper = require('casper').create({ verbose : true ,
    logLevel : 'debug'
})
casper.log('this is a debug message' , 'debug'); casper.log('and an informative one' , 'info');
casper.log('and a warning' , 'warning'); casper.log('and an error' , 'error'); casper.exit();
```

This will give the following output:



The screenshot shows a terminal window titled "2. niko@n1k0: ~/Sites/casperjs (zsh)". The command run was "casperjs log-colors.js". The output displays four log entries with different colors: a green "[debug]" level message, a blue "[info]" level message, a magenta "[warning]" level message, and a red "[error]" level message. The terminal window has a dark background and light-colored text.

```
2. niko@n1k0: ~/Sites/casperjs (zsh)
..ites/casperjs (zsh)
$ casperjs log-colors.js
[debug] [phantom] this is a debug message
[info] [phantom] and an informative one
[warning] [phantom] and a warning
[error] [phantom] and an error
$
```

Fig. 9.1: image

Hint: CasperJS doesn't write logs on the filesystem. You have to implement this by yourself if needed.

CHAPTER 10

Extending

Sometimes it can be convenient to add your own methods to a Casper object instance; you can easily do so as illustrated in the example below:

```
var casper = require('casper').create({ verbose : true ,  
    logLevel : "debug"  
});  
  
var links = {  
    'http://edition.cnn.com/' : 0 ,  
    'http://www.nytimes.com/' : 0 ,  
    'http://www.bbc.co.uk/' : 0 ,  
    'http://www.guardian.co.uk/' : 0  
};  
  
casper.countLinks = function () {  
    return this . evaluate( function () {  
        return __utils__.findAll('a[href]').length;  
    } );  
};  
  
casper.renderJSON = function ( what ) {  
    return this . echo(JSON.stringify(what, null , '' ));  
};  
  
casper.start();  
  
casper.each( Object . keys(links), function ( casper, link ) {  
    this . thenOpen(link, function () {  
        links[link] = this . countLinks();  
    });  
};  
  
casper.run( function () {  
    this . renderJSON(links).exit();  
});
```

```
});
```

But that's just plain old *monkey-patching* the casper object, and you may probably want a more OO approach... That's where the `inherits()` function from the `utils` module and ported from `nodejs` comes handy:

```
var Casper = require('casper').Casper;
var utils = require('utils');
var links = {
    'http://edition.cnn.com/' : 0,
    'http://www.nytimes.com/' : 0,
    'http://www.bbc.co.uk/' : 0,
    'http://www.guardian.co.uk/' : 0
};

function Fantomas() {
    Fantomas.super_.apply(this, arguments);

    // Let's make our Fantomas class extending the Casper one // please note that at this point, CHILD CLASS PROTOTYPE WILL BE
    // OVERRIDDEN
    utils.inherits(Fantomas, Casper);

    Fantomas.prototype.countLinks = function () {
        return this.evaluate(function () {
            return __utils__.findAll('a[href]').length;
        });
    };

    Fantomas.prototype.renderJSON = function (what) {
        return this.echo(JSON.stringify(what, null, ' '));
    };
}

var fantomas = new Fantomas({
    verbose : true,
    logLevel : "debug"
});

fantomas.start();

Object.keys(links).forEach(function (url) {
    fantomas.thenOpen(url, function () {
        links[url] = this.countLinks();
    });
});

fantomas.run(function () {
    this.renderJSON(links).exit();
});
```

Note: The use of the `super_` child class property which becomes available once its parent has been defined using `inherits();` it contains a reference to the parent constructor.

Don't forget to call "Casper"'s parent constructor!

Of course this approach is bit more verbose than the easy *monkey-patching* one, so please ensure you're not just overengineering stuff by subclassing the Casper class.

Using CoffeeScript

If you're writing your casper scripts using [CoffeeScript](#), extending casper is getting a bit more straightforward:

```
links =
  'http://edition.cnn.com/' : 0
  'http://www.nytimes.com/' : 0
  'http://www.bbc.co.uk/' : 0
  'http://www.guardian.co.uk/' : 0

class Fantomas extends require('casper').Casper
  countLinks: ->
    @evaluate ->
      __utils__.findAll('a').length

  renderJSON: (what) ->
    @echo JSON.stringify what, null, ''

fantomas = new Fantomas
  loadImages: false
  logLevel: "debug"
  verbose: true

fantomas.start()

for url of links
  do (url) ->
    fantomas.thenOpen url, ->
      links[url] = @countLinks()

fantomas.run ->
  @renderJSON links @exit()
```


CHAPTER 11

Debugging

A few tips for debugging your casper scripts:

- *Use the verbose mode*
- *Hook in the deep using events*
- *Dump serialized values to the console*
- *Localize yourself in modules*
- *Name your closures*

Use the verbose mode

By default & by design, a Casper instance won't print anything to the console. This can be very limiting & frustrating when creating or debugging scripts, so a good practice is to always start coding a script using these settings:

```
var casper = require('casper').create({ verbose : true ,  
    logLevel : "debug"  
});
```

The verbose setting will tell Casper to write every logged message at the logLevel logging level onto the standard output, so you'll be able to trace every step made.

Warning: Output will then be pretty verbose, and will potentially display sensitive informations onto the console.
Use with care on production.

Hook in the deep using events

Events are a very powerful features of CasperJS, and you should probably give it a look if you haven't already. Some interesting events you may eventually use to debug your scripts:

- The `http.status.XXX` event will be emitted everytime a resource is sent with the `HTTP` code corresponding to `XXX`;
- The `remote.alert` everytime an `alert()` call is performed client-side;
- `remote.message` everytime a message is sent to the client-side console;
- `step.added` everytime a step is added to the stack;
- etc. . .

Listening to an event is dead easy:

```
casper.on('http.status.404', function (resource) {
    this.log('Hey, this one is 404: ' + resource.url, 'warning');
});
```

Ensure to check the [full list](#) of all the other available events.

Dump serialized values to the console

Sometimes it's helpful to inspect a variable, especially Object contents. The `utils_dump()` function can achieve just that:

```
require('utils').dump({ foo : {
    bar : 42
},});
```

Note: `utils_dump()` won't be able to serialize function nor complex cyclic structures though.

Localize yourself in modules

Warning: Deprecated since version 1.1. As of 1.1, CasperJS uses PhantomJS' builtin `require` and won't expose the `__file__` variable anymore.

If you're creating Casper modules, a cool thing to know is that there's a special built-in variable available in every module, `__file__`, which contains the absolute path to current javascript file (the module file).

Name your closures

Probably one of the most easy but effective best practice, always name your closures:

Hard to track:

```
casper.start('http://foo.bar/' , function () {
    this . evaluate( function () {
        // ...
   }); });
});
```

Easier:

```
casper.start('http://foo.bar/' , function afterStart() {
    this . evaluate( function evaluateStuffAfterStart() {
        // ...
   }); });
});
```

That way, everytime one is failing, its name will be printed out in the stack trace, so you can more easily locate it within your code.

Note: This one also applies for all your other Javascript works, of course ;)

CHAPTER 12

FAQ

Here's a selection of the most frequently asked questions by CasperJS newcomers:

- *Is CasperJS a node.js library?*
- *I'm stuck! I think there's a bug! What can I do?*
- *The casper.test property is undefined, I can't write any test!*
- *I keep copy and pasting stuff in my test scripts, that's boring*
- *What is the versioning policy of CasperJS?*
- *Can I use jQuery with CasperJS?*
- *Can I use CasperJS without using the casperjs executable?*
- *How can I catch HTTP 404 and other status codes?*
- *Where does CasperJS write its logfile?*
- *What's this mysterious __utils__ object?*
- *How does then() and the step stack work?*
- *I'm having hard times downloading files using download()*
- *Is it possible to achieve parallel browsing using CasperJS?*
- *Can I access & manipulate DOM elements directly from the CasperJS environment?*
- *Why can't I create a new casper instance in a test environment?*
- *Okay, honestly, I'm stuck with Javascript.*
- *How do I use PhantomJS page module API in casperjs?*
- *How do I provide my implementation of a remote resource?*
- *I'm getting intermittent test failure, what can I do to fix them?*

Is CasperJS a node.js library?

No. CasperJS is written on top of [PhantomJS](#), which is a node-independent [Qt / WebKit](#) based library. If you try to run your CasperJS script with node, it just won't work out of the box.

Hint: If you want to drive CasperJS from node, try [SpookyJS](#).

I'm stuck! I think there's a bug! What can I do?

Before rage-tweeting:

1. Read the [docs](#)
2. Check if an [issue](#) has been open about your problem already
3. Check you're running the [latest stable tag](#)
4. Check you're running the [latest version of PhantomJS](#)
5. Ask on the project mailing list :
 - (a) try to post a reproducible, minimal test case
 - (b) compare casperjs results with native phantomjs ones
 - (c) if the problem also occurs with native phantomjs, ask on [phantomjs mailing list](#)
6. Eventually, file an issue .

The `casper.test` property is undefined, I can't write any test!

That's because as of 1.1, the `casper.test` property is only set to a [`Tester`](#) instance when using the `casperjs test` subcommand. You may want to read the [testing documentation](#) for more information.

I keep copy and pasting stuff in my test scripts, that's boring

Have a look at [this gist](#), it might help. Also, don't forget that CasperJS supports a [CommonJS-compliant module pattern implementation](#).

Note: CasperJS' implementation of `require()` differs a bit from the one provided by [PhantomJS](#), but I personally never encountered any functional difference.

What is the versioning policy of CasperJS?

Releases will follow the [SemVer standard](#); they will be numbered with the follow format:

```
<major>.<minor>.<patch>[-<identifier>]
```

And constructed with the following guidelines:

- Breaking backwards compatibility bumps the major
- New additions without breaking backwards compatibility bumps the minor
- Bug fixes and misc changes bump the patch
- Unstable, special and trunk versions will have a proper identifier

Can I use jQuery with CasperJS?

Sure, you can use [jQuery](#), as every single other javascript library on Earth. A first solution is to inject it into the remote DOM environment by hand using the standard `WebPage.injectJs()` method:

```
casper.page.injectJs('path/to/jquery.js');
```

In the event that you require jQuery being available on every page, you can make use of the `clientScripts` option of CasperJS:

```
var casper = require('casper').create({ clientScripts: [ "includes/jquery.min.js" ]});
```

Note: You can't *inject* scripts using the HTTP protocol, you actually have to use a relative/absolute filesystem path to the script resource.

Can I use CasperJS without using the casperjs executable?

Yes, you can call a CasperJS script directly with the phantomjs executable, but if you do so, you must set the `phantom.casperPath` property to the path where the library root is located on your system:

```
// casperscript.js
phantom.casperPath = '/path/to/casperjs'; phantom.injectJs(phantom.casperPath + '/bin/bootstrap.js');

var casper = require('casper').create();
// ...
```

You can run such a script like any other standard [PhantomJS](#) script:

```
$ phantomjs casperscript.js
```

If you're on Windows, this is the way you may manage to get casper working the most easily:

```
phantom.casperPath = 'C:\path\to\your\repo\lib\casperjs-0.6.X'; phantom.injectJs(phantom.casperPath + '\bin\bootstrap.js');

var casper = require('casper').create();
```

```
// do stuff
```

How can I catch HTTP 404 and other status codes?

You can define your own HTTP status code handlers by using the `httpStatusHandlers` option of the Casper object. You can also catch other HTTP status codes as well, as demoed below:

```
var casper = require('casper').create();

casper.on('http.status.404', function (resource) {
    this.echo('wait, this url is 404: ' + resource.url);
});

casper.on('http.status.500', function (resource) {
    this.echo('woops, 500 error: ' + resource.url);
});

casper.start('http://mywebsite/404', function () {
    this.echo('We suppose this url return an HTTP 404');
});

casper.thenOpen('http://mywebsite/500', function () {
    this.echo('We suppose this url return an HTTP 500');
});

casper.run(function () {
    this.echo('Done.')
});
```

Hint: Check out all the other cool *events* you may use as well.

Where does CasperJS write its logfile?

Nowhere. CasperJS doesn't write logs on the filesystem. You have to implement this by yourself if needed.

What's this mysterious `__utils__` object?

The `__utils__` object is actually a [ClientUtils object](#) which have been automatically injected into the page DOM and is therefore always available. So everytime to perform an `evaluate()` call, you have this instance available to perform common operation like:

- fetching nodes using CSS3 or XPath selectors,
- retrieving information about element properties (attributes, size, bounds, etc.),
- sending AJAX requests,
- triggering DOM events

Check out the [whole API](#). You even have [a bookmarklet](#) to play around with this `__utils__` instance right within your browser console!

Note: You're not obliged at all to use the `__utils__` instance in your scripts. It's just there because it's used by CasperJS internals.

How does `then()` and the step stack work?

Disclaimer This entry is based on an [answer I made on Stack Overflow](#). The `then()` method basically adds a new navigation step in a stack. A step is a javascript function which can do two different things:

1. waiting for the previous step - if any - being executed
2. waiting for a requested url and related page to load

Let's take a simple navigation scenario:

```
var casper = require('casper').create();

casper.start();

casper.then(function step1() {
    this.echo('this is step one');
});

casper.then(function step2() {
    this.echo('this is step two');
});

casper.thenOpen('http://google.com', function step3() {
    this.echo('this is step 3 (google.com is loaded)');
});
```

You can print out all the created steps within the stack like this:

```
require('utils').dump(casper.steps.map(function (step) {
    return step.toString();
}));
```

That gives:

```
$ casperjs test -steps.js [
    "function step1() { this.echo('this is step one'); }",
    "function step2() { this.echo('this is step two'); }",
    "function _step() { this.open(location, settings); }",
    "function step3() { this.echo('this is step 3 (google.com is loaded)'); }"
]
```

Notice the `_step()` function which has been added automatically by CasperJS to load the url for us; when the url is loaded, the next step available in the stack —which is `step3()` — is `then` called. When you have defined your navigation steps, `run()` executes them one by one sequentially:

```
casper.run();
```

Note: The callback/listener stuff is an implementation of the [Promise pattern](#).

I'm having hard times downloading files using download()

You should try to disable *web security*. Using the `-- web-security` command line option:

```
$ casperjs --web-security=no myscript.js
```

Within code:

```
var casper = require('casper').create({ pageSettings : {  
    webSecurityEnabled : false  
}});
```

Or anytime:

```
casper.page.settings.webSecurityEnabled = false ;
```

Is it possible to achieve parallel browsing using CasperJS?

Officially no, but you may want to try.

Can I access & manipulate DOM elements directly from the CasperJS environment?

No. Like in PhantomJS, you have to use [`Casper#evaluate\(\)`](#) to access actual page DOM and manipulate elements. For example, you can't do this:

```
// this won't work  
casper.then( function () {  
    var titleNode = document.querySelector('h1');  
    this.echo('Title is: ' + titleNode.textContent); titleNode.textContent = 'New title';  
  
    this.echo('Title is now: ' + titleNode.textContent);});
```

You have to use the [`Casper#evaluate\(\)`](#) method in order to communicate with the page DOM:

```
// this will  
casper.then( function () {  
    var titleText = this.evaluate(function () {  
        return document.querySelector('h1').textContent;  
    });
```

```
this . echo( 'Title is: ' + titleText);
this . evaluate( function () {
    document .querySelector( 'h1' ).textContent = 'New title' ; });

this . echo( 'Title is now: ' + this . evaluate( function () {
    return document .querySelector( 'h1' ).textContent;
}); );
```

Of course, it's a whole lot more verbose, but Casper provides convenient methods to ease accessing elements properties, eg. `Casper#fetchText()` and `Casper#getElementInfo()`:

```
// this will
casper.then( function () {
    this . echo('Title is: ' + this . fetchText( 'h1' ));
    this . evaluate( function () {
        document .querySelector( 'h1' ).textContent = 'New title' ; });

    this . echo( 'Element HTML is now: ' + this . getElementInfo( 'h1' ).html);
});
```

Why can't I create a new `casper` instance in a test environment?

The `casperjs test subcommand` is a convenient utility which bootstraps and configures a `test environment` for you, so a preconfigured `casper` object is already available in your test script when using this command. As of 1.1-beta3, you're prevented from overriding this preconfigured instance as this practice prevents the test runner from working properly. If you try to create a new casper instance in a test script, you'll get an error and CasperJS will exit with an error message with a link pointing to the documentation.

One may argue this is mostly related to some historical bad design decisions, and this might be true. This behavior is not likely to exist anymore in a future 2.0.

Okay, honestly, I'm stuck with Javascript.

Don't worry, you're not alone. Javascript is a great language, but it's far more difficult to master than one might expect at first look.

Here are some great resources to get started efficiently with the language:

- Learn and practice Javascript online at [Code Academy](#)
- [Eloquent Javascript](#)
- [JavaScript Enlightenment \(PDF\)](#)
- last, a [great tutorial on Advanced Javascript Techniques](#) by John Resig, the author of jQuery. If you master this one, you're almost done with the language.

How do I use PhantomJS page module API in casperjs?

After `casperjs.start()`, you have `phantomjs` page module available in `casper.page` (<http://docs.casperjs.org/en/latest/modules/casper.html#page>)

You can simply do like below:

```
casper.page.nameOfMethod()
```

PhantomJS Web Page API: <http://phantomjs.org/api/webpage/>

How do I provide my implementation of a remote resource?

Using phantomjs native `onResourceRequested` event, you can override remote resource url to your own implementation. Your own implementation file can be provided from local path too:

```
casper.page.onResourceRequested = function (requestData, networkRequest) {
    var match = requestData.url.match(/wordfamily.js/g);
    if (match != null) {
        console.log('Request #' + requestData.id + ': ' + JSON.
.. stringify(requestData));

        // overrides wordfamily.js to local newWordFamily.js
        networkRequest.changeUrl('newWordFamily.js');
    }
};
```

I'm getting intermittent test failure, what can I do to fix them?

This is probably because you are executing a test before the resource or element is available and the page is fully loaded/rendered. This can even happen on things like modals and dynamic content. You can solve this problem by using the `wait*` operations:

```
casper.thenOpen(url, function initialAppearance() {
    casper.waitForText('Text in deep part of page or modal');
});
```

It is good practice to wait for DOM nodes, text, or resources before beginning your tests. It will help make them stable and predictable while still running fast.

CHAPTER 13

Cookbook

This is a collection of scripts and ideas that aim to solve common situations that are encountered by users. This is by no means an exhaustive list, and we encourage you to contribute your recipes on [github](#).

Creating a web service

Warning: It is worth noting that this is probably not the best of ideas. You should be careful of things like memory leaks, lack of long term stability (due to said leaks), and the overall memory hog that headless JS can be.

With the above caveat in mind, a web service would look something like:

```
//filename: server.js

//define ip and port to web service
var ip_server = '127.0.0.1:8585';

//includes web server modules
var server = require('webserver').create();

//start web server
var service = server.listen(ip_server, function (request, response) {
    var links = [];
    var casper = require('casper').create();

    function getLinks() {
        var links = document.querySelectorAll('h3.r a');
        return Array.prototype.map.call(links, function (e) {
            return e.getAttribute('href')
        });
    }
});
```

```
casper.start('http://google.com', function () {
    // search for 'casperjs' from google form
    this.fill('form[action="/search"]', { q : request.postRaw }, true );
});

casper.then(function () {
    // aggregate results for the 'casperjs' search
    links = this.evaluate(getLinks);
});

casper.run(function () {
    response.statusCode = 200;

    //sends results as JSON object
    response.write(JSON.stringify(links, null, null));
    response.close();
});

console.log('Server running at http://' + ip_server + '/');

```

You can start the server by executing:

```
casperjs server.js
```

You can then access the results via an HTTP POST request:

```
curl --data "casperjs" http://127.0.0.1:8585/
```

The above command would search for “casperjs” on google and return a JSON array of results. This is a trivial example and can be expanded into something more complex.

Script to automatically check a page for 404 and 500 errors

```
var casper = require("casper").create({ pageSettings : {

    loadImages : false ,
    loadPlugins : false
});
var checked = [];
var currentLink = 0;
var fs = require('fs');
var upTo = ~~ casper.cli.get('max-depth') || 100;
var url = casper.cli.get(0);
var baseUrl = url;
var links = [url];
var utils = require('utils');
var f = utils.format;

function absPath(url, base) {
    return new URL(url).resolve( new URI(base)).toString();
}

// Clean links
function cleanLinks(urls, base) {
```

```

return utils.unique(urls).filter( function ( url ) {
    return url.indexOf(baseUrl) === 0 || ! new RegExp( '^(\#|ftp|javascript|http)' ).test(url); }).map( function ( url ) {

    return absPath(url, base); }).filter( function ( url ) {

    return checked.indexOf(url) === - 1 ; });

// Opens the page, perform tests and fetch next links
function crawl(link) {
    this . start().then( function () {
        this . echo(link, 'COMMENT' );
        this . open(link); checked.push(link);
    });

    this . then( function () {
        if ( this . currentHttpStatus === 404 ) {
            this . warn(link + ' is missing (HTTP 404)' );
        } else if ( this . currentHttpStatus === 500 ) {
            this . warn(link + ' is broken (HTTP 500)' );
        } else {
            this . echo(link + f( 'is okay (HTTP %s)', this . currentHttpStatus)); } });

    this . then( function () {
        var newLinks = searchLinks.call( this );
        links = links.concat(newLinks).filter( function ( url ) {
            return checked.indexOf(url) === - 1 });

        this . echo(newLinks.length + " new links found on " + link); } );

    // Fetch all <a> elements from the page and return // the ones which contains a href starting with 'http://'
}

function searchLinks() {
    return cleanLinks( this . evaluate( function _fetchInternalLinks() {
        return [].map.call(__utils__.findAll('a[href)'), function ( node ) {
            return node.getAttribute('href');
       }); }), this . getCurrentUrl()); }

// As long as it has a next link, and is under the maximum limit, will keep running
function check() {
    if ( links[currentLink] && currentLink < upTo ) {
        crawl.call( this , links[currentLink]); currentLink ++;

        this . run(check); } else {

    this . echo( "All done, " + checked.length + " links checked." );
    this . exit(); } }

if ( ! url) {
}

```

```
casper.warn('No url passed, aborting.').exit();
}

casper.start('https://gist.githubusercontent.com/pieplu/
.. 6be55d1d8f27ea9b8dcf4de6b1933547/raw/5e8d996fe1d86edac93825d0050fd359caf74f8e/URI.js
.. function () {
    var scriptCode = this . getPageContent() +'; return URI;';
    window .URI = new Function (scriptCode)();
    if ( typeof window .URI === "function" ) {
        this . echo( 'URI.js loaded' );
    } else {
        this . warn( 'Could not setup URI.js' ).exit();
    }
}).then( function () {
    this . echo( "Starting" );
}).run(check);
```

Run it with:

```
casperjs 404checker.js http://mysite.tld/ [-max-depth=42]
```

[Reference gist](#).

Test drag&drop

Assuming a page containing a draggable element like that `one`, we can test drag&drop that way:

```
casper.test.begin('Test drag&drop', 2, function ( test ) {
    casper.start('http://localhost:8000/example.html', function () {
        test.assertEval( function () {
            var pos = $('#box').position();
            return ( pos.left == 0 && pos.top == 0 );
        }, "The box is at the top" );
        this . mouse.down( 5 , 5 );
        this . mouse.move( 400 , 200 );
        this . mouse.up( 400 , 200 );
    });
    casper.then( function () {
        test.assertEval( function () {
            var pos = $('#box').position();
            return ( pos.left == 395 && pos.top == 195 );
        }, "The box has been moved" );
    });
    casper.run( function () {
        test.done();
    });
});
```

Passing parameters into your tests

Let's say you want to be able to change the Uri your tests visits depending on what you are testing. To do this, you can add custom `-parameter=value` to your cli.

```
// casperjs test /foo/bar --url=test.html
var url = 'http://localhost:8000'
var cli = casper.cli

if (cli.has('url')) { url = cli.get('url')

}

console.log( 'InitUsing url: ' + url + '\n' )

casper.test.begin(...)
```

You can find the complete documentation for the `cli` object in <http://docs.casperjs.org/en/latest/cli.html>

CHAPTER 14

Changelog

The CasperJS changelog is [hosted on github](#).

CHAPTER 15

Upgrading

Upgrading to 1.1

Testing framework refactor

The most visible change is the way you write tests. With 1.0, you were able to access a `.test` property from any casper script and so running a suite using the standard `casperjs` executable:

```
// 1.0 style test script not using the 'casperjs test' subcommand
var casper = require('casper').create();

casper.start('http://foo.bar', function () {
    this .test.assert(true );
});

casper.run(function () {
    this .test.done( 1 );
    this .test.renderResults(true );
});
```

In 1.1, the test framework has been heavily refactored to decouple the tester from a casper instance as much as possible, so it's no more possible to run a test suite right from the standard `casperjs` command as you would have done with the script shown above. Instead you now have to use the `casperjs test` subcommand mandatorily to access a tester instance from the `casper.test` property.

Warning: As of 1.1:

- you shouldn't invoke the `renderResults()` method directly anymore
- you shouldn't use the `done()` first argument to set planned test as it's been deprecated
- you can't access the `casper.test` property when not using the `casperjs test` subcommand

If you try, you'll get an error:

```
// test.js
var casper = require('casper').create(); casper.test.assert(true);
```

Will give:

```
$ casperjs test.js
CasperError: casper.test property is only available using the `casperjs test` command
```



The new Tester#begin() method

However, a new `begin()` method has been added to the `Tester` prototype, to ease describing your tests:

```
casper.test.begin('Description of my test', 1, function (test) {
    test.assert(true);
    test.done();
});
```

More asynchronously:

```
casper.test.begin('Description of my test', 1, function (test) {
    casper.start('http://foo.bar', function () {
        test.assert(true);
    });

    casper.run(function () {
        test.done();
    });
});
```

Note: Please notice `begin()`'s second argument which is now the place to set the number of planned tests.

require() in custom modules

CasperJS 1.1 now internally uses PhantomJS' native `require()` function, but it has side effect if you write your own casperjs modules; in any casperjs module, you now have to use the new global `patchRequire()` function first:

```
// casperjs module code
var require = patchRequire(require);
// now you can require casperjs builtins
var utils = require('utils'); exports = {

    // ...
};
```

Note: You don't have to use `patchRequire()` in a standard casperjs script.

__file__ has been removed

As of 1.1, CasperJS now uses native PhantomJS' require() function which doesn't support the `__file__` builtin variable within custom modules like 1.0 allowed.

Tester#getFailures() and Tester#getPasses() methods removed

These two methods have been removed from the `Tester` API.

You can retrieve test failure and success records by simply accessing `tester.currentSuite.failures` and `tester.currentSuite.passes` instead.

Step and run completion callbacks don't throw anymore

Instead, you should listen to the `step.error` and `complete.error` events; if you really want to keep raising them:

```
casper.on( "step.error complete.error" , function ( error ) {
    throw error;});
```


CHAPTER 16

Known Issues

This is a non-exhaustive list of issues that the CasperJS team is aware of and tracking.

PhantomJS

Versions below 2.0.0:

- [phantomjs-issue-10795](#):

There is a known issue while doing clicks within the page that causes execution to halt. It has been fixed in v2.0.0+ in phantomjs.

It is mentioned in the following issues: #[233](#)

```
console.log('START click'); console.log( document.getElementById('foo').toString()); console.log( document.getElementById('foo').click()); // this ends execution

console.log('END click'); // this never gets called
```

Version 2.0.0:

- [phantomjs-issue-12506](#):

`Webpage.uploadFile` is not working. It has been fixed in v2.0.1+ in phantomjs.

- [phantomjs-issue-12410](#):

Quote from PhantomJS 2.0 Release Note :

"PhantomJS 2 can not run scripts written in CoffeeScript anymore (see issue [12410](#)). As a workaround, CoffeeScript users can still compile their scripts to JavaScript first before executing it with PhantomJS."

CHAPTER 17

Credits

Author

CasperJS is mainly developed by [Nicolas Perriault](#) on its free time.

If you want to thank him and/or sponsor the development of CasperJS, please consider donating (see links in the sidebar).

Contributors

These people have contributed to CasperJS:

- Brikou CARRE
- Thomas Parisot
- Han Yu
- Chris Lorenzo
- Victor Yap
- Rob Barreca
- Tyler Ritchie
- Nick Rabinowitz
- Pascal Borreli
- Dave Lee
- Andrew Childs
- Solomon White
- Reina Sweet

- Jan Schaumann
- Elmar Langholz
- Clochix
- Donovan Hutchinson
- Julien Moulin
- Michael Geers
- Jason Funk
- Vladimir Chizhov
- Jean-Philippe Serafin
- snkashis
- Rafael
- Andrew de Andrade
- Ben Lowery
- Chris Winters
- Christophe Benz
- Harrison Reiser
- Jan Pochyla
- Jan-Martin Fruehwacht
- Julian Gruber
- Justin Slattery
- Justine Tunney
- KaroDidi
- Leandro Boscariol
- Maisons du monde
- Marcel Duran
- Mathieu Agopian
- Mehdi Kabab
- Mikko Peltonen
- Rafael Garcia
- Raphael Benitte
- Tim Bunce

Logo

CasperJS logo designed by [Jeremy Forveille](#)

You can download the logo sources here:

- logo CasperJS (PDF)
- logo CasperJS (EPS)
- logo CasperJS (AI) These assets are

under MIT license

CHAPTER 18

License

CasperJS is released under the terms of the [MIT license](#).

Copyright (c) 2011 - {{year}} Nicolas Perriault 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. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

You can also search the genindex if you're looking for something particular.

CHAPTER 19

Community

- [get the code and contribute](#)
- [join the mailing list](#)
- [check out the ecosystem](#)
- [follow @casperjs_org on Twitter](#)
- there's also a [Google+ account](#) (not much updated though)

Symbols

`_utils_`, 69 , 136

A

AJAX, 76 , 136

alert, 63

arguments, 10

Asynchronicity, 56 , 62 , 65 , 92 , 94 , 137

auth, 29 , 53 , 109

B

Base64, 31 , 34 , 70 , 72

Binary, 72

bookmarklet, 70

Browser testing, 18

Bugs, 6 , 127 , 134

bypass, 32 , 57

C

Casper, 25

Casper options, 25

Child Process, 63

CLI, 10

click, 32 , 33 , 107 , 110

Client scripts, 26

Client utils, 69

Code reuse, 134

coffeescript, 9 , 126

colorizer, 77

Colors, 77 , 94

Command line, 10

Community, 159

Continuous Integration, 21

Contributing, 134 , 159

Cookbook, 140

CORS, 138

CSS, 15

CSS3, 15

Custom module, 102

D

Debugging, 36 , 47 , 70 , 99 , 127

DOM, 14 , 38 , 39 , 43 – 45 , 48 , 61 , 69 – 71 , 83 , 84 , 88 , 92 , 102

DOMReady, 57

done(), 94

download, 37 , 107 , 138

dump, 99

E

echo, 38 , 70

error, 26 , 27 , 88 , 108 , 134

Error handling, 27 , 28

evaluate, 38 , 83

EventEmitter, 54

events, 48 , 105 , 129 , 130

Examples, 140

exec File, 63

exit, 26 , 39 , 108

extending, 22 , 124

F

falsiness, 85 , 101

FAQ, 131

fileDownload, 108

fill, 108

filters, 119

Form, 41 , 46 , 52 , 74 , 85

Frames, 68

Framesets, 68

Functional testing, 18

G

git, 4

Globals, 46

H

Help, 131 , 159

Helpers, 99

Homebrew, 3
HTML, 14
HTTP, 26, 27, 49, 51, 65, 86, 89, 109, 111, 112, 136
HTTP Headers, 49
HTTP Method, 49
HTTP Request, 49
HTTP Response, 56
HTTP Status Code, 86

I
Iframes, 68
inheritance, 100, 124
initialization, 54
Installation, 1
InstanceOf, 91

J
Jenkins, 21
jQuery, 70, 135
JSON, 99, 102

K
Known Issues, 151

L
Licensing, 157
log, 75, 110, 136
log levels, 12, 121
Logging, 12, 26, 30, 40, 121, 136

M
Memory, 35, 36
Modules, 102
modules, 23
Mouse, 80

N
New window, 64, 68
Node.js, 133

O
options, 10, 20, 25

P
PhantomJS, 3, 10, 29
planned tests, 92
Popups, 64, 68
Printing, 38
Printing styles, 78
prototype, 124
Python, 3, 135

R
Raw values, 13

Remote scripts, 29
REPL, 6
Ruby, 135
run, 51

S
Samples, 140
screenshot, 33 – 35, 107, 119
Scroll, 52
selector, 14, 66, 88
Serialization, 99
settings, 29, 129
setUp, 20
Shell, 10
sleep, 61
SlimerJS, 3, 12
Spawn, 63
SSL, 29
stack trace, 131
start, 54
Step stack, 28, 30, 32, 51, 56, 57, 137
String formatting, 99
Support, 159

T
Tabs, 64, 68
tearDown, 20
Termination, 92, 94
Test failure, 95
Test success, 97
Test suite, 18, 92, 94
Testing, 16, 82, 134
timeout, 28, 30, 31
truthiness, 90, 101
Type, 91

U
Unit testing, 17
URL, 43, 66, 91
User Agent, 60
Utilities, 99

V
verbose, 30, 123, 129
Versionning, 134
viewport, 30, 61, 118

W
wait, 61
Web security, 138
window, 46
window.open, 64, 68
Windows, 5, 78, 135

X

[XML](#), 21
[XPath](#), 16 , 74
[XSS](#), 29
[XUnit](#), 21