# Phantom.js

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

## About the Tutorial

PhantomJS is a lightweight headless browser built on WebKit. It is called headless because the execution does not happen on the browser, but on the terminal.

This tutorial covers most of the topics required for a basic understanding of PhantomJS. Additionally, this tutorial also explains how to deal with its various components and to get a feel of how it works.

## Audience

This tutorial is designed for those programmers who want to learn the basics of PhantomJS and its programming concepts. It will give you enough understanding on various functionalities of PhantomJS with suitable examples.

## Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of HTML, CSS, JavaScript, and Document Object Model (DOM).

## Copyright and Disclaimer

# Table of Contents

# 1. PHANTOMJS – OVERVIEW

**PhantomJS** is said to be a **headless browser** because there is no involvement of browser while executing the JavaScript code. The execution will not be seen in the browser display, but on the command line prompt. The functionalities like **CSS Handling**, **DOM Manipulation**, **JSON**, **Ajax**, **Canvas**, **SVG**, etc., will all be taken care at the command prompt. PhantomJS does not have a GUI and hence, all its execution takes place at the command line.

Using PhantomJS, we can write to a file, read the contents of the file or upload a file, take an screen capture, convert the webpage into a pdf and lots more. With headless browsers, you get all the browser behavior i.e. **Cookies**, **Http Request Methods** i.e. **GET /POST**, **Clearing Cookies**, **Deleting Cookies**, etc., **Reloading of Page**, **Navigating to a Different Page**.

PhantomJS uses WebKit that has a similar browsing environment like the famous browsers – Google Chrome, Mozilla Firefox, Safari, etc. It also provides a lot of JavaScript API, which helps in taking screenshots, uploading of file, writing to file, reading a file, interacting with the web pages, etc.

PhantomJS does not support Flash or Video, as it requires plugins and it is difficult to support the same on the command line.

## Features of PhantomJS

Let us now understand the features that PhantomJS offers.

### Page Automation

PhantomJS allows you to read the page contents with the help of its API. It can help to manipulate the DOM, use external libraries to carry out the task required.

### Screen Capture

PhantomJS helps in taking a screen capture of a page specified and save the same as an image in various formats like PNG, JPEG, PDF, and GIF. With the help of the screen capture, it can easily help to make sure the web content is fine.

PhantomJS offers properties and methods with the help of which it allows developers to adjust the size of the screenshots and specify the area they want to capture.

### Headless Testing

PhantomJS helps testing of UI at the command line. While, with the help of a screenshot, it can easily help to find errors in the UI. PhantomJS sometimes cannot help with testing alone. However, it can be wrapped along with other testing libraries like Mocha, Yoeman, etc. You can take the help of PhantomJS to upload a file and submit the form.

PhantomJS can be used to test logins across the sites and make sure the security is not compromised. PhantomJS can be used along with other tools like **CasperJS**, **Mocha**, **Qunit** to make the testing more powerful.

## Network Monitoring

One of the important features of PhantomJS is its usage to monitor the network connection using the API available. PhantomJS permits the inspection of network traffic; it is suitable to build various analysis on the network behavior and performance.

PhantomJS can be used to collect the data about the performance of the webpage in a live environment. PhantomJS can be used with tools like **Yslow** to gather performance metrics of any websites.

# 2. PHANTOMJS – ENVIRONMENT SETUP

PhantomJS is a free software and is distributed under the **BSD License**. It is easy to install and it offers multiple features to execute the scripts. PhantomJS can be easily run on multiple platforms such as Windows, Linux, and Mac.

For downloading PhantomJS, you can go to – http://phantomjs.org/ and then click on the download option.

## For Windows

The download page shows you the options for download for different OS. Download the zip file, unpack it and you will get an executable **phantom.exe**. Set the PATH environment variable to the path of phantom.exe file. Open a new command prompt and type **phantomjs –v**. It should give you the current version of PhantomJS that is running.

## For MAC OS X

Download the PhantomJS zip file meant for MAC OS and extract the content. Once the content is downloaded, move the PhantomJS to – **/usr/local/bin/**. Execute PhantomJS command i.e. phantomjs –v at the terminal and it should give you the version description of PhantomJS.

## Linux 64 bit

Download the PhantomJS zip file meant for Linux 64 bit and extract the content. Once the content is downloaded, move PhantomJS folder to **/usr/local/share/** and create a **symlink**:

```
sudo mv $PHANTOM_JS /usr/local/share
sudo ln -sf /usr/local/share/$PHANTOM_JS/bin/phantomjs /usr/local/bin.
```

Execute phantomjs –v at the terminal and it should give the version of PhantomJS.

## Linux 32 bit

Download the PhantomJS zip file meant for Linux 32 bit and extract the content. Once the content is downloaded, move the PhantomJS folder to **/usr/local/share/** and create a symlink:

```
sudo mv $PHANTOM_JS /usr/local/share
sudo ln -sf /usr/local/share/$PHANTOM_JS/bin/phantomjs /usr/local/bin.
```

Execute phantomjs –v at the terminal and it should give the version of PhantomJS.

The PhantomJS source code can also be taken from the git repository by clicking on the following link – https://github.com/ariya/phantomjs/

To run scripts in PhantomJS, the command is as follows:

```
phantomjs jsfile arg1 arg2…
```

# 3. PHANTOMJS – OBJECT

In this chapter, we will look at the four important objects PhantomJS. They are as follows:

- CookiesEnabled
- Cookies
- LibraryPath
- Version

Let us now discuss each of these in detail.

## cookiesEnabled

It tells whether the cookies are enabled or not. It will return **true**, if yes; otherwise **false**.

### Syntax

Its syntax is as follows:

```
phantom.cookiesEnabled
```

### Example

**cookieenabled.js**

```
phantom.addCookie({
  //adding cookie with addcookie property
  name: 'c1',
  value: '1',
  domain: 'localhost'
});
console.log("Cookie Enabled value is : "+phantom.cookiesEnabled);
phantom.exit();
```

### Output

**Command:** phantomjs cookieenabled.js

```
Cookie Enabled value is : true
```

# Cookies

It helps to add and set cookies to a domain. It returns an object with all the cookies available for the domain.

## Syntax

Its syntax is as follows:

```
phantom.cookies;
```

## Example

**Filename: phantomcookie.js**

```
phantom.addCookie({
  name: 'c1',
  value: '1',
  domain: 'localhost'
});
phantom.addCookie({
  name: 'c2',
  value: '2',
  domain: 'localhost'
});
phantom.addCookie({
  name: 'c3',
  value: '3',
  domain: 'localhost'
});
console.log(JSON.stringify(phantom.cookies));
phantom.exit();
```

## Output

**Command:** phantomjs phantomcookie.js

```
[{"domain":".localhost","httponly":false,"name":"c3","path":"/","secure":false,"
value":"3"},{"domain":".localhost","httponly":false,"name":"c2","path":"/","secu
re":false,"value":"2"},{"domain":".localhost","httponly":false,"name":"c1","path
":"/","secure":false,"value":"1"}]
```

In the above example, we added some cookies to the localhost domain. We then fetched it using **phantom.cookies**. It returns an object with all the cookies by using the **JSON stringify** method to convert the JavaScript object into a string. You can also use **foreach** to access the name/values of the cookies.

# LibraryPath

PhantomJS libraryPath stores the script path to be used by the **injectJS** method.

## Syntax

Its syntax is as follows:

```
phantom.libraryPath
```

## Example

Here is an example to find out the version.

```
var webPage = require('webpage');
var page = webPage.create();


page.open('http://www.tutorialspoint.com/jquery', function(status) {
  if (status === "success") {

page.includeJs('http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js',
function() {
      if (page.injectJs('do.js')) {
        var title = page.evaluate(function() {
          // returnTitle is a function loaded from our do.js file - see below
          return returnTitle();
        });
        console.log(title);
        phantom.exit();
      }
    });
  }
});
window.returnTitle = function() {
  return document.title;
};
```

The above program generates the following **output**.

```
Jquery Tutorial
```

# Version

It gives the version of the PhantomJS that is running and returns the details in an object. For example: {"major":2,"minor":1,"patch":1}

## Syntax

Its syntax is as follows:

```
phantom.version
```

## Example

Here is an example to find out the version.

```
var a = phantom.version;
console.log(JSON.stringify(a));
console.log(a.major);
console.log(a.minor);
console.log(a.patch);
phantom.exit();
```

The above program generates the following **output**.

```
{"major":2,"minor":1,"patch":1}
2
1
1
```

In the above example, we have used **console.log** to print the version. Currently, we are running on version 2. It returns the object with the details shown in the above code block.

# 4. PHANTOMJS – METHODS

PhantomJS is a platform to help execute JavaScript without a browser. To do that, the following methods are used, which help in Adding the Cookie, Deleting, Clearing, Exiting the Script, Injecting JS, etc.

We will discuss more on these PhantomJS methods and their syntax in this chapter. Similar methods i.e. **addcookie**, **injectjs** exists on the webpage module, which will be discussed in the subsequent chapters.

PhantomJS exposes the following methods that can help us to execute JavaScript without the browser:

- addCookie
- clearCookie
- deleteCookie
- Exit
- InjectJS

Let us now understand these methods in detail with examples.

## addCookie

The addcookie method is used to add cookies and store in the data. It is similar to how the browser stores it. It takes a single argument that is an object with all the properties of cookies and the syntax for it looks like shown below:

### Syntax

Its syntax is as follows:

```
phantom.addCookie({
"name" : "cookie_name",
"value" : "cookie_value",
"domain" : "localhost"
});
```

The name, value, domain are mandatory properties to be added to the addcookie function. If any of this property is missing in the cookie objects, this method will fail.

- **name:** specifies the name of the cookie.
- **value:** specifies the value of the cookie to be used.
- **domain:** domain to which the cookie will be applied.

## Example

Here is an example of the **addcookie** method.

```
var page = require('webpage').create(),url  = 'http://localhost/tasks/a.html';
page.open(url, function(status) {
  if (status === 'success') {
    phantom.addCookie({  //add name cookie1 with value =1
      name: 'cookie1',
      value: '1',
      domain: 'localhost'
    });
     phantom.addCookie({ // add cookie2 with value 2
      name: 'cookie2',
      value: '2',
      domain: 'localhost'
    });
    phantom.addCookie({ // add cookie3 with value 3
      name: 'cookie3',
      value: '3',
      domain: 'localhost'
    });
    console.log('Added 3 cookies');
    console.log('Total cookies :'+phantom.cookies.length);
    // will output the total cookies added to the url.
    } else {
     console.error('Cannot open file');
    phantom.exit(1);
  }
});
```

## Example

**a.html**

```
<html>
<head><title>Welcome to phantomjs test page</title></head>
<body>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
</body>
</html>
```

The above program generates the following **output**.

```
Added 3 cookies
Total cookies :3
```

The code comments are self-explanatory.

## clearCookies

This method allows deleting all the cookies.

### Syntax

Its syntax is as follows:

```
phantom.clearCookies();
```

This concept works similar to deleting the browser cookies by selecting in the browser menu.

### Example

Here is an example of the **clearCookies** method.

```
var page = require('webpage').create(),url  = 'http://localhost/tasks/a.html';
page.open(url, function(status) {
  if (status === 'success') {
    phantom.addCookie({  //add name cookie1 with value =1
```

```
      name: 'cookie1',
      value: '1',
      domain: 'localhost'
    });
     phantom.addCookie({ // add cookie2 with value 2
      name: 'cookie2',
      value: '2',
      domain: 'localhost'
    });
    phantom.addCookie({ // add cookie3 with value 3
      name: 'cookie3',
      value: '3',
      domain: 'localhost'
    });
    console.log('Added 3 cookies');
    console.log('Total cookies :'+phantom.cookies.length);
    phantom.clearCookies();
    console.log('After clearcookies method total cookies :'
+phantom.cookies.length);
    phantom.exit();
  } else {
    console.error('Cannot open file');
    phantom.exit(1);
  }
});
```

## a.html

```
<html>
<head><title>Welcome to phantomjs test page</title></head>
<body>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
```

```
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
</body>
</html>
```

The above program generates the following **output**.

```
Added 3 cookies
Total cookies :3
After clearcookies method total cookies :0
```

## deleteCookie

Delete any cookie in the **CookieJar** with a 'name' property matching cookieName. It will return **true**, if successfully deleted; otherwise **false**.

### Syntax

Its syntax is as follows:

```
phantom.deleteCookie(cookiename);
```

Let us understand **addcookie**, **clearcookies** and **deletecookie** with the help of an example.

### Example

Here is an example to demonstrate the use of deleteCookie method:

**File: cookie.js**

```
var page = require('webpage').create(),url  = 'http://localhost/tasks/a.html';
page.open(url, function(status) {
  if (status === 'success') {
    phantom.addCookie({  //add name cookie1 with value =1
      name: 'cookie1',
      value: '1',
      domain: 'localhost'
    });
     phantom.addCookie({ // add cookie2 with value 2
      name: 'cookie2',
```

```
      value: '2',
      domain: 'localhost'
    });
    phantom.addCookie({ // add cookie3 with value 3
      name: 'cookie3',
      value: '3',
      domain: 'localhost'
    });

    console.log('Added 3 cookies');
    console.log('Total cookies :'+phantom.cookies.length); //will output the total
cookies added to the url.
    console.log("Deleting cookie2");
    phantom.deleteCookie('cookie2');
    console.log('Total cookies :'+phantom.cookies.length);
    phantom.clearCookies();
    console.log('After clearcookies method total cookies :'
+phantom.cookies.length);
    phantom.exit();
  } else {
    console.error('Cannot open file');
    phantom.exit(1);
  }
});
```

The above program generates the following **output**.

```
phantomjs cookie.js
Added 3 cookies
Total cookies :3
Deleting cookie2
Total cookies :2
After clearcookies method total cookies :0
```

# Exit

The phantom.exit method will exit the script which it had started. It exits the program with return value mentioned. It gives '**0**', if there is no value passed.

## Syntax

Its syntax is as follows:

```
phantom.exit(value);
```

In case you do not add **phantom.exit**, then the command line assumes that the execution is still on and will not complete.

## Example

Let us look at an example to understand the use of the **exit** method.

```
console.log('Welcome to phantomJs');  // outputs Welcome to phantomJS
var a = 1;
if (a === 1) {
  console.log('Exit 1'); //outputs Exit 1
  phantom.exit(); // Code exits.
} else {
  console.log('Exit 2');
  phantom.exit(1);
}
```

The above program generates the following **output**.

**phantomjs exit.js**

```
Welcome to phantomJs
Exit 1
```

Any piece of code after phantom.exit will not be executed, since phantom.exit is a method to end the script.

# injectJs

InjectJs is used to add **addtionaljs** files in phantom. If the file is not found in the current **directory librarypath,** then the phantom property (phantom.libraryPath) is used as an additional place to track the path. It returns **true** if the file addition is successful otherwise **false** for failure, incase if it is not able to locate the file.

## Syntax

Its syntax is as follows:

```
phantom.injectJs(filename);
```

## Example

Let us look at the following example to understand the use of **injectJs**.

**Filename: inject.js**

```
console.log("Added file");
```

**File name: addfile.js**

```
var addfile =  injectJs(inject.js);


console.log(addfile);
phantom.exit();
```

## Output

**Command:** C:\phantomjs\bin>phantomjs addfile.js

```
Added file // coming from inject.js
true
```

In the above example, **addfile.js** calls the file **inject.js** using injectJs. When you execute addfile.js, the console.log present in inject.js is shown in the output. It also shows true for addfile variable since the file inject.js was added successfully.

# Webpage Module

# 5. PHANTOMJS – PROPERTIES

PhantomJS provides quite a lot of properties and methods to help us to interact with the contents inside a webpage.

The require("webpage").create() command creates a webpage object. We will use this object to manipulate the webpage with the help of properties and methods listed below.

```
var wpage = require("webpage").create();
```

The following table has the list of all the webpage properties that we are going to discuss.

| S.No. | Properties & Description |
|---|---|
| 1 | **canGoBack**<br>This property returns **true** if there is previous page in the navigation history; if not, **false**. |
| 2 | **canGoForward**<br>This property returns true if there is next page in the navigation history; if not, **false**. |
| 3 | **clipRect**<br>clipRect is an object with values top, left, width and height and used to take the image capture of the webpage when used by the render() method |
| 4 | **Content**<br>This property contains the contents of webpage. |
| 5 | **cookies**<br>With cookies, you can set /get the cookies available on the URL. It will also give you the cookies available on the URL and the new cookies set on the page. |
| 6 | **customHeaders**<br>customHeaders specifies additional HTTP request headers that will be send to server for every request issued by the page. |
| 7 | **Event**<br>It gives long list of events i.e. modifier, keys details. |

| | |
|---|---|
| 8 | **focusedFrameName**<br>Returns the name of the currently focused frame. |
| 9 | **frameContent**<br>This property gives the content of the frame which is active |
| 10 | **frameName**<br>Returns the name of the currently focused frame. |
| 11 | **framePlainText**<br>This property also gives the contents of the current active frame but only contents without any html tags. |
| 12 | **frameTitle**<br>Gives the title of the active frame. |
| 13 | **frameUrl**<br>This property will give the url of the currently focused frame. |
| 14 | **framesCount**<br>Gives the count of the frames present on the page. |
| 15 | **framesName**<br>Gives array of frame names. |
| 16 | **libraryPath**<br>This property has the path, which is used by page.inectJs method. |
| 17 | **navigationLocked**<br>This property defines whether navigation of the page is allowed or not. If true it will be on current page url and clicking on page to go to next page will not be allowed. |
| 18 | **offlineStoragePath**<br>This property gives the path where the data is stored using window.localStorage.The path can be changed using --local-storage-path from command line. |
| 19 | **offlineStorageQuota**<br>This property defines the maximum amount of data you can store in window.localStorage.The value is 5242880 bytes which is 5MB.This value can overwritten at command line using the following command --local-storage-quota = size over here |

| | |
|---|---|
| 20 | **ownsPages**<br>ownsPages returns true or false if the page opened by the webpage is a child of the webpage. |
| 21 | **pagesWindowName**<br>PagesWindowName will give the names of the windows open using window.open |
| 22 | **pages**<br>The pages property will you give array of pages opened in a page using window.open.If the page is closed in url you referring the page will not be considered. |
| 23 | **paperSize**<br>This property gives the size ie dimensions of the webpage when needs to be used to convert the webpage in a pdf format.paperSize contains the dimensions required in an object |
| 24 | **plaintext**<br>This property also gives the contents of the current active frame but only contents without any html tags. |
| 25 | **scrollPosition**<br>This contains object indicating the scroll position. It gives left and top. |
| 26 | **settings**<br>This property will give the settings of the webpage when page.open method is used. Once the page is loaded the changes in settings properties will not create any impact. |
| 27 | **title**<br>This property will give you the title of the page you are reading |
| 28 | **url**<br>This property will give the page url |
| 29 | **viewportSize**<br>This property allows to change the size of the window display. It contains width and height, which you can read or change it as per the needs. |
| 30 | **windowName**<br>Gives the name of the window. |

| | |
|---|---|
| 31 | **zoomFactor**<br>This property specifies the zoom factor for render and renderBase64 methods. It helps to zoom a page and take a screen capture if required |

## canGoBack

The **canGoBack** property returns **true** if there is a previous page in the navigation history; if not, **false**.

### Syntax

Its syntax is as follows:

```
page.canGoBack;
```

### Example

The following examples demonstrates the use of **canGoBack** property.

```
var wpage = require('webpage').create();
console.log(wpage.canGoBack);
phantom.exit();
```

It generates the following **output**:

```
False
```

## canGoForward

The **canGoForward** property returns true if there is a next page in the navigation history; if not, **false**.

### Syntax

Its syntax is as follows:

```
page.canGoForward;
```

### Example

Let us look at an example of the canGoForward property.

```
var wpage = require('webpage').create();
console.log(wpage.canGoForward);
```

```
phantom.exit();
```

The above program generates the following **output**.

```
False
```

## clipRect

The clipRect is an object with values top, left, width and height and used to take the image capture of the webpage, when used by the **render()** method. If the clipRect is not defined, it will take the screenshot of the full webpage when the render method is called.

### Syntax

Its syntax is as follows:

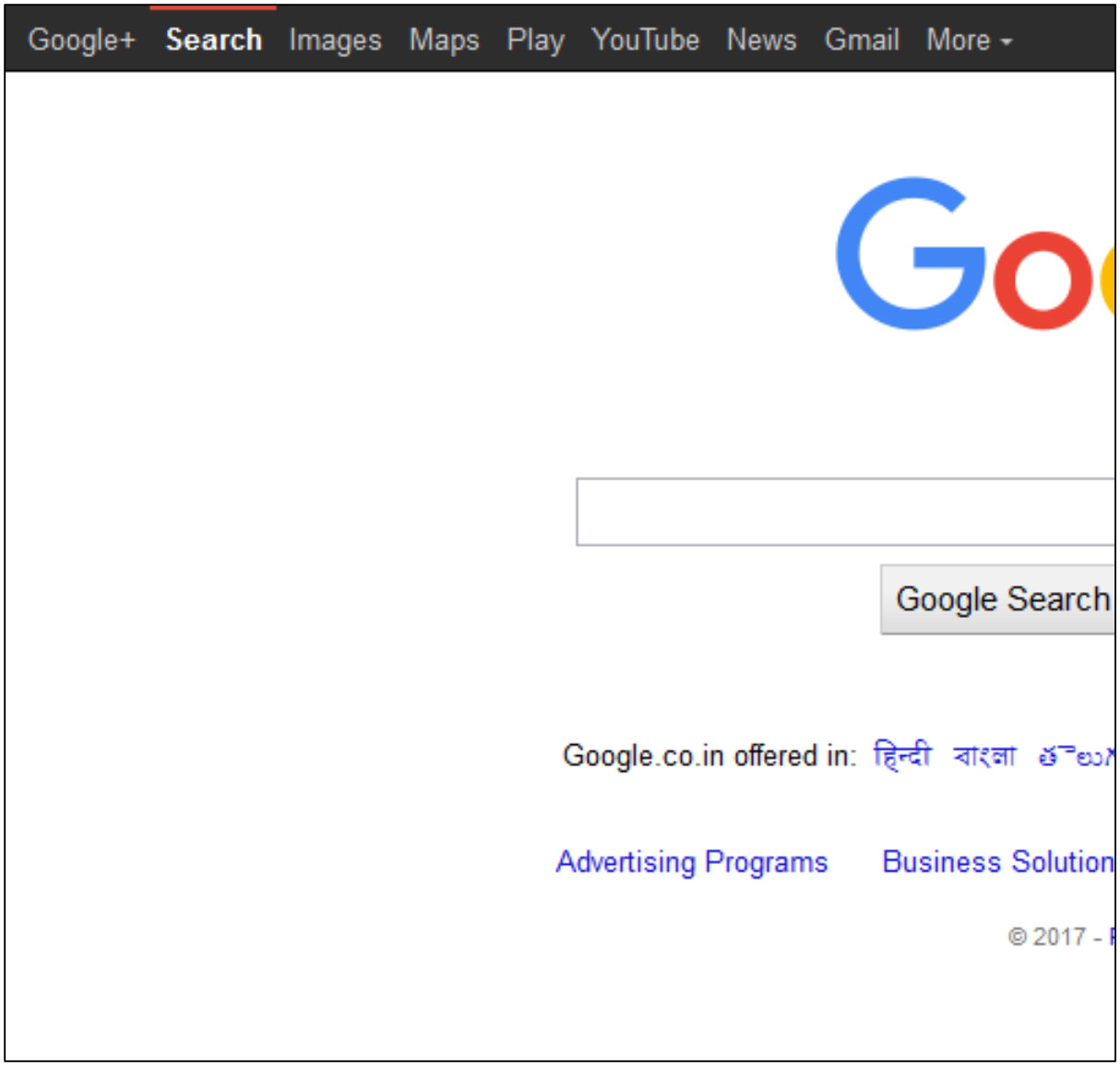
```
var page = require('webpage').create();
page.clipRect = {
  top: 14,
  left: 3,
  width: 400,
  height: 300
};
```

### Example

Take a look at the following example to understand the use of **clipRect** property.

```
var wpage = require('webpage').create();
wpage.viewportSize = { width: 1024, height: 768 };
wpage.clipRect = { top: 0, left: 0, width: 500, height: 500 };
//the clipRect is the portion of the page you are taking a screenshot
wpage.open('http://www.google.com/', function() {
  wpage.render('e.png');
  phantom.exit();
});
```

Here, we are taking the screenshot of the site **google.com**. It will generate the following **output**:




## content

This property contains the contents of a webpage.

### Syntax

Its syntax is as follows:

```
var page = require('webpage').create();
page.content;
```

To show an example let us open up a page and console and see what we get in **page.content**.

The **open webpage method** in detail will be discussed later. Right now, we will use it to explain the properties with it.

## Example

The following example shows how you can use the **content** property.

```
var wpage = require('webpage').create(),url  = 'http://localhost/tasks/a.html';
wpage.open(url, function(status) {
  if (status) {
        console.log(status);
        var content = wpage.content;
        console.log('Content: ' + content);
        phantom.exit();
     } else {
           console.log("could not open the file");
           phantom.exit();
     }
});
```

The above program generates the following **output**.

```
Success
Content: <html><head></head>
<body>
<script type="text/javascript">
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
</script>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
<h1>This is a test page</h1>
```

```
</body></html>
```

Here, we will use the local page to fetch the contents and the output of the page that is shown above. The **page.content** function works just like the **view source** function of the browser.

## cookies

We have cookies property on the phantom object as well as on the **phantom webpage object**. With cookies, you can set /get the cookies available on the URL. It will also give you the cookies available on the URL and the new cookies set on that page.

### Syntax

Its syntax is as follows:

```
page.cookies;
```

### Example

Take a look at the following example to understand how to use the **cookies** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/a.html', function (status) {
  var cookies = wpage.cookies;
  console.log('Cookies available on page are as follows :');
  console.log(JSON.stringify(cookies));
  phantom.exit();
});
```

The above program generates the following **output**.

```
Cookies available on page are as follows :
[{"domain":"localhost","expires":"Fri, 22 Dec 2017 12:00:00 GMT","expiry":151394
4000,"httponly":false,"name":"username","path":"/tasks/","secure":false,"value":
"Roy"}]
```

If you check the **page.content** example, we have set the cookie to the page using document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";

When we try to read the cookies of the page, it lists out all the details of a cookie, such as its Domain, Expires, Httponly, Name, Value, Path, etc. The page.cookies returns all the cookies available on a page.

# customHeaders

The customHeaders function specifies additional HTTP request headers that will be send to server for every request issued by the page. The default value is an empty object "{}". Header name and values are encoded in US-ASCII before being sent to the server.

## Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.customHeaders = {
  //specify the headers
};
```

## Example

The following example shows the use of **customHeaders** property.

```
var page = require('webpage').create();
var server = require('webserver').create();
var port=8080;
var listening = server.listen(8080, function (request, response) {
      console.log("GOT HTTP REQUEST");
      console.log(JSON.stringify(request, null, 4));
});
var url = "http://localhost:" + port + "/foo/response.php";
console.log("sending request to :" +url);
page.customHeaders = {'Accept-Language' : 'en-GB,en-US;q=0.8,en;q=0.6'};
//additional headers are mentioned here
page.open(url, function (status) {
      if (status !== 'success') {
            console.log('page not opening');
      } else {
            console.log("Getting response from the server:");
      }
      phantom.exit();
});
```

## Output with customheader

The above program generates the following output.

```
sending request to :http://localhost:8080/foo/response.php
GOT HTTP REQUEST
{
    "headers": {
        "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "Accept-Encoding": "gzip, deflate",
        "Accept-Language": "en-GB,en-US;q=0.8,en;q=0.6",
        "Connection": "Keep-Alive",
        "Host": "localhost:8080",
        "User-Agent": "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/538.1 (KH
TML, like Gecko) PhantomJS/2.1.1 Safari/538.1"
    },
    "httpVersion": "1.1",
    "method": "GET",
    "url": "/foo/response.php"
}
```

In above example, we are using **customheader** to set the **Accept-Language** and the same is shown in the http headers.

## Output without customheader

The above program generates the following output without the customheader.

```
sending request to :http://localhost:8080/foo/response.php
GOT HTTP REQUEST
{
    "headers": {
        "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "Accept-Encoding": "gzip, deflate",
        "Accept-Language": "en-IN,*",
        "Connection": "Keep-Alive",
        "Host": "localhost:8080",
        "User-Agent": "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/538.1 (KH
```

```
TML, like Gecko) PhantomJS/2.1.1 Safari/538.1"
    },
    "httpVersion": "1.1",
    "method": "GET",
    "url": "/foo/response.php"
}
```

Without the custom header, the accept language is as shown as shown in the above output. Using the customheader property, you can change the required http headers.