



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

© Copyright 2017 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright and Disclaimer	i
Table of Contents	ii
1. PhantomJS – Overview	1
Features of PhantomJS	1
2. PhantomJS – Environment Setup	3
3. PhantomJS – Object	4
cookiesEnabled.....	4
Cookies	5
LibraryPath	6
Version.....	7
4. PhantomJS – Methods	8
addCookie.....	8
clearCookies	10
deleteCookie.....	12
Exit	13
injectJs	14
WEBPAGE MODULE	16
5. PhantomJS – Properties	17
canGoBack	20
canGoForward	20
clipRect	21
content	22
cookies.....	24
customHeaders.....	25
event.....	27
focusedFrameName	28
frameContent	29
frameName.....	30
framePlainText	31
frameTitle	32
frameUrl	34
framesCount.....	34
framesName	36
libraryPath	37
navigationLocked.....	38
offlineStoragePath.....	38
offlineStorageQuota	39
ownsPages	40
pagesWindowName	41
Pages	42
paperSize	43

plainText	46
scrollTop	48
settings	48
title	50
URL	50
viewportSize	51
windowName	52
zoomFactor	53
6. PhantomJS – Methods	55
addCookie ()	58
childFramesCount ()	59
childFramesName ()	59
clearCookies ()	59
close ()	60
currentFrameName ()	61
deleteCookie ()	61
evaluateAsync ()	62
evaluateJavaScript ()	64
evaluate ()	64
getPage ()	65
goBack ()	66
goForward ()	67
go ()	67
includeJs ()	68
injectJs ()	69
openUrl ()	70
Callback	70
open ()	72
release ()	73
reload ()	73
renderBase64 ()	74
renderBuffer ()	75
render ()	75
sendEvent ()	77
setContent ()	79
stop ()	80
switchToFocusedFrame ()	81
switchToFrame ()	83
switchToMainFrame ()	83
switchToParentFrame ()	84
uploadFile ()	84
7. PhantomJS – Events/Callbacks	87
onAlert ()	88
onCallback ()	89
onClosing ()	90
onConfirm ()	91
onConsoleMessage ()	92
onError ()	93
onFilePicker ()	94
onInitialized ()	95

onLoadFinished ()	95
onLoadStarted ()	96
onNavigationRequested ()	96
onPageCreated ()	98
onPrompt ()	99
onResourceError ()	100
onResourceReceived ()	101
onResourceRequested ()	102
onResourceTimeout ()	103
onUrlChanged ()	104
8. PhantomJS – Child Process Module	106
Spawn Method	106
FILE SYSTEM MODULE	108
9. PhantomJS – Properties	109
Separator	109
workingDirectory	109
10. PhantomJS – Methods	111
absolute	113
changeWorkingDirectory	113
copyTree	114
copy	115
exists	116
isAbsolute	116
isDirectory	117
isExecutable	117
isFile	118
isLink	119
isReadable	120
isWritable	120
lastModified	121
list	121
makeDirectory	122
makeTree	123
move	123
open	124
readLink	125
read	126
removeDirectory	127
removeTree	127
remove	128
size	129
touch	130
write	130

SYSTEM MODULE.....	132
11. PhantomJS – Properties	133
args	133
env	134
OS	135
pid.....	135
platform.....	136
WEB SERVER MODULE	137
12. PhantomJS – Properties	138
port.....	138
13. PhantomJS – Methods	139
close.....	139
listen	139
MISCELLANEOUS.....	143
14. PhantomJS – Command Line Interface	144
15. PhantomJS – Screen Capture	146
16. PhantomJS – Page Automation	150
17. PhantomJS – Network Monitoring	151
18. PhantomJS – Testing	164
19. PhantomJS – REPL.....	167
20. PhantomJS – Examples	169

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": "/", "sec
u
re": false, "value": "2"}, {"domain": ".localhost", "httponly": false, "name": "c1", "pat
h
```

```
": "/", "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 **deletetecookie** 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 **additionaljs** 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 This property returns true if there is previous page in the navigation history; if not, false .
2	canGoForward This property returns true if there is next page in the navigation history; if not, false .
3	clipRect 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 <code>render()</code> method
4	Content This property contains the contents of webpage.
5	cookies 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 customHeaders specifies additional HTTP request headers that will be send to server for every request issued by the page.
7	Event It gives long list of events i.e. modifier, keys details.
8	focusedFrameName Returns the name of the currently focused frame.

9	frameContent This property gives the content of the frame which is active
10	frameName Returns the name of the currently focused frame.
11	framePlainText This property also gives the contents of the current active frame but only contents without any html tags.
12	frameTitle Gives the title of the active frame.
13	frameUrl This property will give the url of the currently focused frame.
14	framesCount Gives the count of the frames present on the page.
15	framesName Gives array of frame names.
16	libraryPath This property has the path, which is used by page.inectJs method.
17	navigationLocked 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 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 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 ownsPages returns true or false if the page opened by the webpage is a child of the webpage.

21	pagesWindowName PagesWindowName will give the names of the windows open using window.open
22	pages 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 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 This property also gives the contents of the current active frame but only contents without any html tags.
25	scrollPosition This contains object indicating the scroll position. It gives left and top.
26	settings 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 This property will give you the title of the page you are reading
28	url This property will give the page url
29	viewportSize 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 Gives the name of the window.
31	zoomFactor 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:

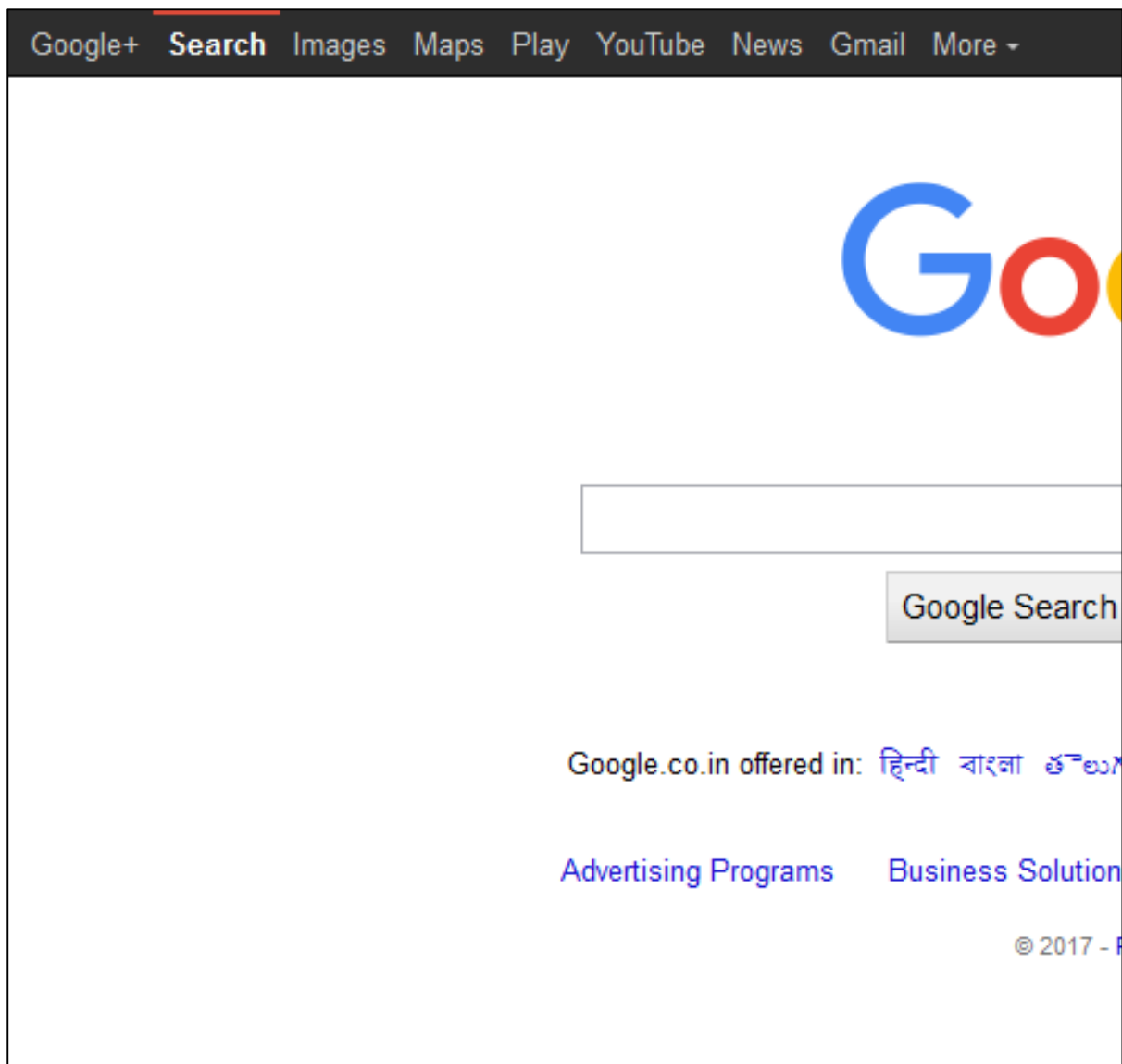
```
var page = require('webpage').create();
page.clipRect = {
  top: 14,
  left: 3,
  width: 400,
  height: 300
};
```

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.

event

It returns a long list of events, i.e., modifier, keys details, etc.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.event;
```

Example

Let us take an example to understand how the **event** property works.

```
var page = require('webpage').create();
console.log(JSON.stringify(page.event));
```

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

```
{ "modifier": { "shift": 33554432, "ctrl": 67108864, "alt": 134217728, "meta": 268435456,
"
keypad": 536870912 }, "key": { "0": 48, "1": 49, "2": 50, "3": 51, "4": 52, "5": 53, "6": 54, "7":
5, "8": 56, "9": 57, "A": 65, "AE": 198, "Aacute": 193, "Acircumflex": 194, "AddFavorite": 16
7
77408, "Adiaeresis": 196, "Agrave": 192, "Alt": 16777251, "AltGr": 16781571, "Ampersand"
:
38, "Any": 32, "Apostrophe": 39, "ApplicationLeft": 16777415, "ApplicationRight": 16777
4
16, "Aring": 197, "AsciiCircum": 94, "AsciiTilde": 126, "Asterisk": 42, "At": 64, "Atilde"
:

```

```
195,"AudioCycleTrack":16777478,"AudioForward":16777474,"AudioRandomPlay":167774
7
6,"AudioRepeat":16777475,"AudioRewind":16777413,"Away":16777464,"B":66,"Back":1
6
777313}}
```

focusedFrameName

The focusedFrameName property returns the name of the currently focused frame.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create()
wpage.focusedFrameName;
```

Example

Let us take a look at an example to understand the use of **focusedFrameName** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/ff.html', function (status) {
    console.log(status);
    console.log(wpage.focusedFrameName);
phantom.exit();
});
```

ff.html

```
<html>
<head><title>welcome to phantomjs</title></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
```

```

        console.log("page is loaded");
    }
</script>

<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<iframe src="http://localhost/tasks/alert.html" width="800" height="800"
name="myframe1" id="myframe1">
</iframe>
<h1>dddddddddd</h1>
</body>
</html>

```

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

```

Success
page2

```

frameContent

This property returns the contents of an active frame. Setting the property will reload the contents of this webpage.

Syntax

Its syntax is as follows:

```

var page = require('webpage').create();
page.frameContent example;

```

Example

Let us take an example to understand the use of **frameContent** property.

```

var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/content.html', function (status) {
    console.log(status);
    console.log(wpage.frameContent); // gives the content of the active frame
    phantom.exit();
});

```

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

```
Success
<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";
window.onload = function() {
    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<h1>dddddddddd</h1>
</body>
</html>
```

frameName

It returns the name of the current active frame.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.frameName;
```

Example

Let us take an example to understand the use of **frameName** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/content.html', function (status) {
    console.log(status);
    console.log(wpage.frameName);
    phantom.exit();
});
```

```
});
```

content.html

```
<html>
<head></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<h1>dddddddddd</h1>
</body>
</html>
```

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

```
success
page2
```

framePlainText

This property gives the content of the current active frame, but only that content which is without any html tags.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.framePlainText;
```

Example

Let us take an example to understand the use of **framePlainText** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/content.html', function (status) {
    console.log(status);
    console.log(wpage.framePlainText);
    phantom.exit();
});
```

content.html

```
<html>
<head><title>welcome to phantomjs</title></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<h1>dddddddddd</h1>
</body>
</html>
```

It will produce the following **output**.

```
success

dddddddddd
```

frameTitle

It returns the title of the active frame.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.frameTitle;
```

Example

Let us take an example to understand the use of **frameTitle** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/content.html', function (status) {
    console.log(status);
    console.log(wpage.frameTitle);
    // this will give me the title of the page in <title></title>
    phantom.exit();
});
```

Content.html

```
<html>
<head><title>welcome to phantomjs</title></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<h1>dddddddddd</h1>
</body>
</html>
```

The above program generates the following output.

```
success  
welcome to phantomjs
```

frameUrl

This property will return the URL of the currently focused frame.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();  
wpage.frameUrl;
```

Example

Let us take an example to understand the use of **frameUrl** property.

```
var wpage = require('webpage').create();  
wpage.open('http://localhost/tasks/content.html', function (status) {  
    console.log(status);  
    console.log(wpage.frameUrl);  
    phantom.exit();  
});
```

The above program generates the following output.

```
success  
http://localhost/tasks/content.html
```

framesCount

It returns the count of the frames present on the page.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();  
wpage.framesCount;
```


Example

Let us look at an example of the framesCount property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/ff.html', function (status) {
    console.log(status);
    console.log(wpage.framesCount);
    phantom.exit();
});
```

ff.html

```
<html>
<head><title>welcome to phantomjs</title></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<iframe src="http://localhost/tasks/alert.html" width="800" height="800"
name="myframe1" id="myframe1">
</iframe>
<h1>dddddddddd</h1>
</body>
</html>
```

The above program generates the following output.

```
Success
2
```

framesName

It returns an array of frame names.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.framesName;
```

Example

Let us take an example to understand the use of **framesName** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/ff.html', function (status) {
    console.log(status);
    console.log(wpage.framesName);
    phantom.exit();
});
```

ff.html

```
<html>
<head><title>welcome to phantomjs</title></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<iframe src="http://localhost/tasks/alert.html" width="800" height="800"
name="myframe1" id="myframe1">
</iframe>
```

```
<h1>dddddddddd</h1>
</body>
</html>
```

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

```
Success
myframe,myframe1
```

libraryPath

This property has the path, which is used by the **page.injectJs** method.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.libraryPath='filepath';
```

Example

Take a look at the following example:

```
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
```

navigationLocked

This property checks whether it is allowed to navigate a page or not. If **true**, it will be on the current page URL and clicking on that page to go to the next page will not be allowed. The default value is false for navigationLocked.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.navigationLocked;
```

Example

Let us take an example to understand the use of **navigationLocked** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/page1.html', function (status) {
    console.log(wpage.navigationLocked);
    phantom.exit();
});
```

It will generate the following **output**.

```
False
```

offlineStoragePath

This property returns the path where the data is stored using the **window.localStorage**. This path can be changed using **--local-storage-path** from the command line.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.offlineStoragePath
```

Example

Let us take an example to understand the use of **offlineStoragePath** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/page1.html', function (status) {
    console.log(wpage.offlineStoragePath );
    phantom.exit();
});
```

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

```
C:/Users/Username/AppData/Local/Ofi Labs/PhantomJS
```

offlineStorageQuota

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 be overwritten at command line using the command **--local-storage-quota = size** over here.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.offlineStorageQuota
```

Example

Let us take an example to understand the use of **offlineStorageQuota** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/page1.html', function (status) {
    console.log(wpage.offlineStorageQuota );
    phantom.exit();
});
```

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

```
5242880
```

ownsPages

The **ownsPages** property checks if a page opened by the webpage is its child page or not. Accordingly, it either returns **true** or **false**.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();  
wpage.ownsPages
```

Example

Let us take an example to understand the use of **ownsPages** property.

```
var wpage = require('webpage').create();  
wpage.open('http://localhost/tasks/page1.html', function (status) {  
    console.log(wpage.ownsPages);  
    phantom.exit();  
});
```

page1.html

```
<html>  
<head>  
<title>Testing PhantomJs</title>  
</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";  
    window.onload = function() {  
        console.log("page is loaded");  
        window.open("http://localhost/tasks/a.html", "page1");  
    }  
</script>  
  
<h1>This is a test page</h1>  
</body>  
</html>
```

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

```
True
```

pagesWindowName

The `pagesWindowName` property returns the names of the windows that are opened using **window.open**.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();  
wpage.pagesWindowName;
```

Example

Let us take an example to understand the use of **pagesWindowName** property.

```
var wpage = require('webpage').create();  
wpage.open('http://localhost/tasks/ptitle.html', function (status) {  
    console.log(wpage. pagesWindowName);  
    phantom.exit();  
});
```

ptitle.html

```
<html>  
<head>  
<title>Testing PhantomJs</title>  
</head>  
<body>  
<script type="text/javascript">  
window.onload = function() {  
    window.open("http://localhost/tasks/a.html", "page1");  
    window.open("http://localhost/tasks/content.html", "page2");  
}  
</script>  
<h1>This is a test page</h1>
```

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

It will generate the following **output**.

```
page1, page2
```

The output gives an array of page names opened using the window.open command. If the window is closed, it is not considered.

Pages

The **Pages** property will you give an array of pages opened in a page using window.open. If the page is closed in the URL you referred, the page will not be considered.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.pages;
```

Example

Let us take an example to understand the use of **page** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/ptitle.html', function (status) {
    console.log(wpage.pages);
    phantom.exit();
});
```

ptitle.html

```
<html>
<head>
<title>Testing PhantomJs</title>
</head>
<body>
<script type="text/javascript">
window.onload = function() {
    window.open("http://localhost/tasks/a.html", "page1");
    window.open("http://localhost/tasks/content.html", "page2");
```



```

}
</script>

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

```

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

```
WebPage(name = "WebPage"),WebPage(name = "WebPage")
```

The webpage we are referring in the above example i.e. **phtml.html** has two window.open commands. The output shows an array of pages from **wpages.pages**.

paperSize

This property gives the size i.e. dimensions of the webpage, which needs to be used to convert the webpage in a pdf format. The paperSize property contains the dimensions required in an object. If paperSize is not defined, the size will be taken of the webpage. Dimensions units supported are 'mm', 'cm', 'in' and 'px'. By default, it is 'px'.

Parameters

Following are the parameters of the paperSize property

- **Margin:** It can be given as an object with values 'top','left','bottom','right'.By default it will considered as 0. For example – margin: {top: '100px',left: '20px',right: '20px',bottom: '10px'}
- **Format:** Formats supported are 'A3','A4','A5','Legal','Letter','Tabloid'.
- **Orientation:** 'Portrait' and 'Landscape'.By default it is 'Portrait'.
- **Headers and Footers:** Header and footer can be provided in an object format with height and contents property.

Syntax

Its syntax is as follows:

```

header: {
  height: "1cm",
  contents: phantom.callback(function(pageNumber, nPages) {
    return "<h1>Header <b>" + pageNumber + " / " + nPages + "</b></h1>";
  })
}
footer: {

```

```

    height: "1cm",
    contents: phantom.callback(function(pageNumber, nPages) {
        return "<h1>Footer <b>" + pageNumber + " / " + nPages + "</b></h1>";
    })
}

```

The syntax for **paperSize** is as follows:

```

wpage.paperSize = {
    width: '600px',
    height: '1500px',
    margin: {'top': '50px',

    'left': '50px',
    'right': '50px'
    },
    orientation: 'portrait',
    header: {
        height: "1cm",
        contents: phantom.callback(function(pageNumber, nPages) {
            return "<h5>Header <b>" + pageNumber + " / " + nPages + "</b></h5>";
        })
    },
    footer: {
        height: "1cm",
        contents: phantom.callback(function(pageNumber, nPages) {
            return "<h5>Footer <b>" + pageNumber + " / " + nPages +
            "</b></h5>";
        })
    }
}

```

Example

Let us take an example to understand the use of **paperSize** property.

```

var wpage = require('webpage').create();
var url = "http://localhost/tasks/a.html";
var output = "test.pdf";

```

```

wpage.paperSize = {
  width: '600px',
  height: '1500px',
  margin: {'top': '50px',
    'left': '50px',
    'right': '50px'
  },
  orientation: 'portrait',
  header: {
    height: "1cm",
    contents: phantom.callback(function(pageNumber, nPages) {
      return "<h5>Header <b>" + pageNumber + " / " + nPages + "</b></h5>";
    })
  },
  footer: {
    height: "1cm",

    contents: phantom.callback(function(pageNumber, nPages) {
      return "<h5>Footer <b>" + pageNumber + " / " + nPages +
"</b></h5>";
    })
  }
}

wpage.open(url, function (status) {
  if (status !== 'success') {
    console.log('Page is not opening');
    phantom.exit();
  } else {
    wpage.render(output);
    phantom.exit();
  }
});

```

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

```
test.pdf
```

In the above example, we are opening a URL and giving papersize options to it. The `wpage.render (output)` converts the URL given to pdf format. The pdf file will be stored in the output mentioned, which in the above example we have given as `var output = "test.pdf"`.

We can define the location where you want to store the file. It gives you the pdf format with the papersize dimension used with the header and the footer. You can execute the above code and see how the pdf files are rendered.

plainText

The **plainText** property returns the contents of the webpage as plain text without any html tags in it.

Syntax

Its syntax is as follows:

```
wpage.plainText
```

Example

Let us take an example to understand the use of **plainText** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/a.html', function (status) {
    console.log(wpage.plainText);
    phantom.exit();
});
```

a.html

```
<html>
<head></head>
<body name="a">
<script type="text/javascript">
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
    console.log("page is loaded");
}
```

```
</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>
```

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

```
This is a test page  
  
This is a test page  
  
This is a test page  
  
This is a test page  
  
This is a test page  
  
This is a test page  
  
This is a test page  
  
This is a test page
```

The `plainText` property just returns the contents without any script tags or html tags.

scrollTop

This contains object indicating the scroll position. It gives left and top. You can change the values for left and top to change the scrollposition of the page. Alternatively, just read the value to know the page scroll position.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create()
wpage.scrollTop;
```

Example

Let us take an example to understand the use of **scrollTop** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/page1.html', function (status) {
    console.log(JSON.stringify(wpage.scrollTop));
    wpage.scrollTop = {
        top:500,
        left:30
    }
    console.log(JSON.stringify(wpage.scrollTop));
    phantom.exit();
});
```

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

```
{"left":0,"top":0}
{"left":30,"top":500}
```

settings

This property will give the settings of the webpage when page.open method is used. Once the page is loaded, the changes in the settings properties will not create any impact. It allows you to read /change the required settings.

Following are the values stored in the settings objects:

- **XSSAuditingEnabled**: False. It has default value to false and it defines whether the load request should be monitored for cross-domains scripts.
- **javascriptCanCloseWindows**: True. To activate/deactivate closing of windows opened from a page.
- **javascriptCanOpenWindows**: True. To activate/deactivate opening of windows from a page.
- **javascriptEnabled**: True. To enable/disable javascript. By default, it is true.
- **loadImages**: True. To activate/deactivate loading of images. By default, it is set to true.
- **localToRemoteUrlAccessEnabled**: True. It defines whether locally if remote URL's can be accessed or not. By default, it is true.
- **userAgent**: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/538.1. Gives the details of the userAgent when the pages are requested from the server.
- **webSecurityEnabled**: True. Defines if security of web should be enabled or not. It is true by default.
- **resourceTimeout**: (in milli-secs) defines the timeout after which any resource requested will stop trying and proceed with other parts of the page.onResourceTimeout callback will be called on timeout.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.settings
```

Example

Let us take an example to understand the use of **settings** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/page1.html', function (status) {
    console.log(JSON.stringify(wpage.settings));
    phantom.exit();
});
```

It will generate the following **output**.

```
{"XSSAuditingEnabled":false,"javascriptCanCloseWindows":true,"javascriptCanOpenW
```

```
indows":true,"javascriptEnabled":true,"loadImages":true,"localToRemoteUrlAccess
E
nabled":false,"userAgent":"Mozilla/5.0 (Windows NT 6.2; WOW64)
AppleWebKit/538.1
(KHTML, like Gecko) PhantomJS/2.1.1 Safari/538.1","webSecurityEnabled":true}
```

title

This property returns the title of the page you are reading.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.title;
```

Example

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

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/ptitle.html', function (status) {
    console.log(wpage.title);
    phantom.exit();
});
```

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

```
Testing PhantomJs
```

URL

This property returns the current page URL.

Syntax

Its syntax is as follows:

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



```
wpage.url;
```

Example

The following example shows the use of URL property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/ptitle.html', function (status) {
    console.log(wpage.url);
    phantom.exit();
});
```

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

```
http://localhost/tasks/ptitle.html
```

viewportSize

This property allows changing the size of the window display. It contains width and height, which you can read or change as per the requirement.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.viewportSize;
```

Example

Let us take an example to understand the use of **viewportSize** property.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/page1.html', function (status) {
    console.log(JSON.stringify(wpage.viewportSize));
    wpage.viewportSize = {
        'width' : 800,
        'height' : 800
    }
});
```

```

        console.log(JSON.stringify(wpage.viewportSize));
        phantom.exit();
    });

```

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

```

{"height":300,"width":400}
{"height":800,"width":800}

```

windowName

This property returns the name of the window.

Syntax

Its syntax is as follows:

```

var wpage = require('webpage').create();
wpage.windowName;

```

Example

Let us take an example to understand the use of **windowName** property.

```

var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/ff.html', function (status) {
    console.log(status);
    console.log(wpage.windowName);
    phantom.exit();
});

```

ff.html

```

<html>
<head><title>welcome to phantomjs</title></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');

```

```
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<iframe src="http://localhost/tasks/alert.html" width="800" height="800"
name="myframe1" id="myframe1">
</iframe>
<h1>dddddddddd</h1>
</body>
</html>
```

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

```
success
page2
```

zoomFactor

This property specifies the zoom factor for **render** and **renderBase64** methods. It helps to zoom a page and take a screen capture if required. Value higher than 1 will increase the size of the page and less than 1 decreases.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.zoomFactor;
```

Example

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

```
var wpage = require('webpage').create();
wpage.zoomFactor = 0.3;

wpage.open("http://www.google.com", function (status) {
    window.setTimeout(function () {
```

```
wpage.render('output.png');  
phantom.exit();  
}, 200);  
});
```

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



6. PhantomJS – Methods

The Web Page Module has methods for **Cookies, Frames, Page Navigation, Reload, Rendering** and **Uploading of Files**.

Following are the methods available on the web page.

S.No.	Methods & Description
1	addCookie() addCookie method adds cookies to the page specified.
2	childFramesCount() This method is deprecated as per http://phantomjs.org .
3	childFramesName() This method is deprecated as per http://phantomjs.org .
4	clearCookies() Will delete all the cookies for the page specified.
5	close() This method is used to close the page and release the memory used. Any of the webpage methods or properties will not work once the close is called.
6	currentFrameName() This method is deprecated as per http://phantomjs.org
7	deleteCookie() This will delete a cookie with the name matching with the existing list of cookies present for a given page url.
8	evaluateAsync() Evaluate given function asynchronously within the page without blocking current execution. This function helps to execute certain scripts asynchronously.
9	evaluateJavascript() EvaluateJavaScript helps to execute the function passed to it as a string. Please note the string passed has to be a function only.

10	evaluate() Evaluate will execute the function passed to it. If the function contain console messages it is not displayed directly in the terminal. To display any console messages you need to use <code>onConsoleMessage</code> phantom callback.
11	getPage() This will give you the child page that matches the windowname passed in <code>getPage</code> .
12	goBack() It gives the previous page in navigation history, if only the navigation is not locked.
13	goForward() It gives the next page in navigation history, if only the navigation is not locked.
14	go() This method allows you to navigate with the pages.
15	includeJS() Includejs executes the external JS file on page and executes the callback function on completion.
16	injectjs() InjectJs includes external script from a specified file into the page. If the file is not available in the current directory, then it used <code>libraryPath</code> for additional search of the file. It returns true if the file is injected, otherwise false.
17	openUrl() OpenUrl opens up a webpage. It is similar to open method of PhantomJS. OpenUrl has some additional parameters, which are <code>httpConf</code> , settings and callback functions.
18	open() Open is used to open up a webpage.
19	release() Release will release the memory heap usage for the page. Do not use page instance methods after the release is called. This method is similar to the close method and the usage of it is deprecated. Instead use <code>wpage.close()</code> .
20	reload() Reload is used to reload the current page used.

21	renderBase64() This method takes the screen capture and gives the image as a string in base46. Renderbase64 supports format like PNG, JPEG and JPG. It does not support gif as of now. You can use clipRect property to define the portion for image capture.
22	renderBuffer() RenderBuffer takes the capture of the webpage to an image buffer, which can be directly sent to the server. Formats supported are PNG, GIF and JPEG.
23	render() Render helps to take the image buffer and save it as per the format specified.
24	sendEvent() It is used to send an event to the webpage. They are not dom events. Each of these events are sent to the webpage based on user interaction.
25	setContent() setContent will change the page content of the specified url with the contents specified.
26	stop() It helps to stop loading of the page.
27	switchToChildFrame() It is deprecated to use switchToFrame();
28	switchToFocusedFrame() It selects the frame, which is in focus.
29	switchToFrame() Selects frame with the name specified and which is child of current frame.
30	switchToMainFrame() Selects mainframe i.e. the root window.
31	switchToParentFrame() It takes the parent frame of the current child frame.
32	uploadFile() This method is used to handle the file upload done using form in html. PhantomJS does not have a direct way to do so using forms the same can be achieved using uploadFile method. It takes html tag selector for file location and the destination where it has to be copied.

addCookie ()

The addCookie method adds cookies to the page specified. For the cookie to be added, the domain name has to match the page otherwise, the cookie is ignored. It returns true, if added successfully otherwise false. The **Name**, **Value** and **Domain** are mandatory fields in the addcookie method.

Right now, we will add cookies to the page **a.html**. Therefore, wpage.cookies will give the newly added cookie and the existing cookies present on page a.html.

Syntax

Its syntax is as follows:

```
phantom.addCookie({
  'name'      : 'cookie1',    /* mandatory property */
  'value'     : '1234',      /* mandatory property */
  'domain'    : 'localhost',  /* mandatory property */
  'path'      : '/',
  'httponly'  : true,
  'secure'    : false,
  'expires'   : (new Date()).getTime() + (5000 * 60 * 60)
});
```

Example

Let us look at an example of the addCookie () method.

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

phantom.addCookie({
  'name'      : 'cookie1',    /* mandatory property */
  'value'     : '1234',      /* mandatory property */
  'domain'    : 'localhost',  /* mandatory property */
  'path'      : '/',
  'httponly'  : true,
  'secure'    : false,
  'expires'   : (new Date()).getTime() + (5000 * 60 * 60)
});

wpage.open('http://localhost/tasks/a.html', function() {
  console.log(JSON.stringify(wpage.cookies));
});
```



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

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

```
[{"domain": ".localhost", "expires": "Sun, 07 May 2017 01:13:45 GMT", "expiry": 149499825, "httponly": true, "name": "cookie1", "path": "/", "secure": false, "value": "1234"}, {"domain": "localhost", "expires": "Fri, 22 Dec 2017 12:00:00 GMT", "expiry": 151394000, "httponly": false, "name": "username", "path": "/tasks/", "secure": false, "value": "Roy"}]
```

childFramesCount ()

This method is deprecated as per <http://phantomjs.org>.

childFramesName ()

This method is deprecated as per <http://phantomjs.org>.

clearCookies ()

The ClearCookies () method will delete all the cookies for the page specified.

Syntax

Its syntax is as follows:

```
wpage.clearCookies();
```

Example

Let us look at an example of the clearCookies () method.

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

phantom.addCookie({
  'name'      : 'cookie1',      /* mandatory property */
  'value'     : '1234',        /* mandatory property */
  'domain'    : 'localhost',    /* mandatory property */
  'path'      : '/',
  'httponly'  : true,
  'secure'    : false,
```

```

    'expires' : (new Date()).getTime() + (5000 * 60 * 60)
  });

wpage.open('http://localhost/tasks/a.html', function() {
    console.log("Cookies available are :");
    console.log(JSON.stringify(wpage.cookies));
    console.log("Clearing all cookies");
    wpage.clearCookies();
    console.log("Cookies available now");
    console.log(JSON.stringify(wpage.cookies));
    phantom.exit();
});

```

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

```

Cookies available are :
[{"domain": ".localhost", "expires": "Sun, 07 May 2017 01:28:18 GMT", "expiry": 1494100698, "httponly": true, "name": "cookie1", "path": "/", "secure": false, "value": "1234"}]
, {"domain": "localhost", "expires": "Fri, 22 Dec 2017 12:00:00 GMT", "expiry": 1513944000, "httponly": false, "name": "username", "path": "/tasks/", "secure": false, "value": "Roy"}]
Clearing all cookies
Cookies available now
[]

```

close ()

The close() method is used to close the page and release the memory used. Any of the webpage methods or properties will not work once the close is called.

Syntax

Its syntax is as follows:

```
wpage.close();
```

Example

The following example shows how to use the **close()** method.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/a.html', function() {
    console.log("page is opened");
    wpage.close();
    phantom.exit();
});
```

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

```
page is opened
```

currentFrameName ()

This method is deprecated as per <http://phantomjs.org>.

deleteCookie ()

The DeleteCookie () method will help to delete a cookie with the name matching with the existing list of cookies present for a given page URL.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.deleteCookie(cookieName);
```

Example

The following example shows the use of **deleteCookie** method.

```
var wpage = require('webpage').create();
phantom.addCookie({
    'name'      : 'cookie1',    /* mandatory property */
    'value'     : '1234',      /* mandatory property */
    'domain'    : 'localhost', /* mandatory property */
    'path'      : '/',
    'httponly'  : true,
    'secure'    : false,
    'expires'   : (new Date()).getTime() + (5000 * 60 * 60)
});
```

```

wpage.open('http://localhost/tasks/a.html', function() {
    console.log("Cookies available are :");
    console.log(JSON.stringify(wpage.cookies));
    wpage.deleteCookie('cookie1');
    console.log("Cookies available now after deleting cookie1");
    console.log(JSON.stringify(wpage.cookies));
    phantom.exit();
});

```

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

```

Cookies available are :
[{"domain": ".localhost", "expires": "Sun, 07 May 2017 10:21:04 GMT", "expiry": 1494132664, "httponly": true, "name": "cookie1", "path": "/", "secure": false, "value": "1234"}
, {"domain": "localhost", "expires": "Fri, 22 Dec 2017 12:00:00 GMT", "expiry": 1513944000, "httponly": false, "name": "username", "path": "/tasks/", "secure": false, "value": "Roy"}]

Cookies available now after deleting cookie1
[{"domain": "localhost", "expires": "Fri, 22 Dec 2017 12:00:00 GMT", "expiry": 1513944000, "httponly": false, "name": "username", "path": "/tasks/", "secure": false, "value": "Roy"}]

```

evaluateAsync ()

This method evaluates the given function asynchronously within the page without blocking current execution. This function helps to execute certain scripts asynchronously.

The **evaluateAsync** method takes arguments as function and second arguments takes time in milliseconds. It is the time taken before the function should execute. This function does not have any return value.

Syntax

Its syntax is as follows:

```
evaluateAsync(function, [delayMillis, arg1, arg2, ...])
```

Example

Let us look at an example of the `evaluateAsync ()` method.

```
var wpage = require('webpage').create();
wpage.onConsoleMessage = function(str) {
    console.log(str);
}
wpage.open("http://localhost/tasks/content.html", function(status) {
    wpage.evaluateAsync(function() {
        console.log('Hi! I\'m evaluateAsync call!');
    }, 1000);
});
```

content.html

```
<html>
<head><title>welcome to phantomjs</title></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {
    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<h1>dddddddddd</h1>
</body>
</html>
```

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

```
welcome to cookie example
```

```
page is loaded  
Hi! I'm evaluateAsync call!
```

evaluateJavaScript ()

The `evaluateJavaScript` method helps to execute the function passed to it as a string. Please note the string passed has to be a function only.

Syntax

Its syntax is as follows:

```
evaluateJavaScript(str);
```

Example

Let us take an example to understand the use of **evaluateJavaScript()** method.

```
var wpage = require('webpage').create();  
wpage.open('http://localhost/tasks/test.html', function(status) {  
    var script1 = "function(){ var a = document.title; return a;}";  
    var value = wpage.evaluateJavaScript(script1);  
    console.log(value);  
    phantom.exit();  
});
```

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

```
Welcome to phantomjs
```

evaluate ()

The **evaluate** method will execute the function passed to it. If the function contains console messages, it is not displayed directly in the terminal. To display any console messages, you need to use `onConsoleMessage` phantom callback.

Syntax

Its syntax is as follows:

```
wpage.evaluate(str)
```

Example

The following example shows how you can use the **evaluate()** method.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/test.html', function(status) {
    var script1 = "function(){ var a = document.title; return a;}";
    var value = wpage.evaluate(script1);
    console.log(value);
    phantom.exit();
});
```

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

```
Welcome to phantomjs
```

Example with Console Messages

Let us consider another example with console messages.

```
var wpage = require('webpage').create();
wpage.onConsoleMessage = function(msg) {
    console.log('CONSOLE: ' + msg);
};
wpage.open('http://localhost/tasks/test.html', function(status) {
    var script1 = "function(){ var a = document.title; console.log('hello world');return a;}";
    var value = wpage.evaluate(script1);
    console.log(value);
    phantom.exit();
});
```

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

```
CONSOLE: hello world
Welcome to phantomjs
```

getPage()

This method returns the child page that matches the windowname passed in the getpage function.

Syntax

Its syntax is as follows:

```
wpage.getPage(windowname)
```

Example

The following example shows the use of **getPage()** method.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/test.html', function(status) {
    var value = wpage.getPage("window2");
    console.log(value);
    phantom.exit();
});
```

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

```
WebPage(name = "WebPage")
```

goBack ()

This function returns the previous page in the navigation history, if only the navigation is not locked.

Syntax

Its syntax is as follows:

```
wpage.goBack()
```

Example

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/test.html', function(status) {
    var value = wpage.goBack();
    console.log(value);
    phantom.exit();
});
```

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

```
False
```


goForward ()

It gives the next page in the navigation history, if only the navigation is not locked.

Syntax

Its syntax is as follows:

```
wpage.goForward()
```

Example

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/test.html', function(status) {
    var value = wpage.goForward();
    console.log(value);
    phantom.exit();
});
```

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

```
False
```

go ()

This method allows you to navigate with the pages. If you want the next/previous page, it is done as follows.

```
wpage.go(1);           // next page
wpage.go(-1);          // previous page
```

Syntax

Its syntax is as follows:

```
wpage.go(index)
```

Example

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/test.html', function(status) {
```

```
var value = wpage.go(-1);
console.log(value);
phantom.exit();
});
```

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

```
False
```

includeJs()

The **includeJs** method executes the external JS file on the page and executes the callback function on completion.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.includeJs(jsfile,function(){});
```

Example

The following example shows the use of **includeJs()** method.

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

wpage.onConsoleMessage = function (str) {
    console.log('CONSOLE: ' + msg);
}

wpage.open('http://localhost/tasks/a.html', function(status) {
    wpage.includeJs('http://localhost/tasks/testscript.js',
        function() {
            var foo = wpage.evaluate(function() {
                return testcode();
            });
            console.log(foo);
        }
    );
});
```

```
});
```

testscript.js

```
function testcode () {
  return "welcome to phantomjs";
}
```

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

```
welcome to phantomjs
```

injectJs ()

The injectJs method includes external script from a specified file into the page. If the file is not available in the current directory, it uses libraryPath for additional search of the file. It returns true, if the file is injected, otherwise false.

Syntax

Its syntax is as follows:

```
wpage.injectJs(filename);
```

Example

The following example shows how to use the injectJs() method.

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/a.html', function(status) {
  if (wpage.injectJs('tscript1.js')) {
    var msg = wpage.evaluate(function() {
      return testcode();
    });
    console.log(msg);
    phantom.exit();
  }
});
```

tscript1.js

```
function testcode () {  
    return "welcome to phantomjs";  
}
```

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

```
welcome to phantomjs
```

openUrl ()

The openUrl method opens up a webpage. It is similar to open method of phantomjs. This method has some additional parameters, which are **httpConf**, **settings** and **callback** functions.

HttpConf

HttpConf is an object that has the following properties:

- **Operation:** It is the http method GET/POST
- **Data:** It is used for POST method.
- **Headers:** An object like wpage.customHeaders.

The default for httpConf is the **get** method. It is optional and you can specify null for the same.

Settings

It is similar to wpage.settings property. You can use null, if you do not want to specify the same.

Callback

It is called when a page is loaded.

Syntax

Its syntax is as follows:

```
wpage = openUrl(url, httpConf, settings);
```

Example

The following example shows the use of **openUrl()** method.

```
var wPage = require('webpage').create();  
wPage.settings.userAgent = 'Mozilla/5.0 (Windows NT 6.1; WOW64)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36';
```

```
wPage.onLoadFinished = function(){
    console.log('Entering on load finished');
    console.log(wPage.content);
    console.log(JSON.stringify(wPage.settings));
}
wPage.openUrl("http://localhost/tasks/a.html","POST", wPage.settings);
```

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

```
Entering on load finished
<html><head><title>Welcome to phantomjs test page</title></head>
<body name="a">
<script type="text/javascript">
    window.onload = function() {
        window.open("http://localhost/tasks/alert.html", "t1");
    }
</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>
{"XSSAuditingEnabled":false,"javascriptCanCloseWindows":true,"javascriptCanOpen
W
indows":true,"javascriptEnabled":true,"loadImages":true,"localToRemoteUrlAccess
E
nabled":false,"userAgent":"Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.3
6 (KHTML, like Gecko) Chrome/37.0.2062.120
Safari/537.36","webSecurityEnabled":t
rue}
```

open ()

The **open** method is used to open up a webpage. The open method takes a page URL and has a callback function, which is called when a page is loaded. The callback function is optional and can be used when required. The callback function contains the status, which defines the success or the failure for the page.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.open(url, function(status) {
    //status is success or failure
});
```

open() with GET method

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

wpage.open('http://www.google.com/', function(status) {
    console.log(status);
    phantom.exit();
});
```

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

```
Success
```

open() with POST method

```
var wpage = require('webpage').create();
var postdata = "username=roy";
wpage.open('http://localhost/tasks/a.php', 'POST', postdata, function(status) {
    console.log(status);
    console.log(wpage.content);
    phantom.exit();
});
```

a.php

```
<?php
```

```
print_r($_POST);
?>
```

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

```
success
<html><head></head><body>Array
(
    [username] => roy
)
</body></html>
```

release ()

The **release** method will release the memory heap usage for the page. Do not use page instance methods after release is called. This method is similar to close and the usage of it is deprecated. Instead, use **wpage.close()**.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.release
```

Example

```
var wpage = require('webpage').create();
wpage.open('http://localhost/tasks/a.html', function() {
    console.log("page is opened");
    wpage.close();
    phantom.exit();
});
```

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

```
page is opened
```

reload ()

The **reload** method is used to reload the current page used.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.reload();
```

Example

```
var wpage = require('webpage').create();
wpage.onLoadFinished = function(status) {
    console.log("Page is loaded");
    setTimeout(function() {
        wpage.reload();
    }, 2000);
};
wpage.open('http://localhost/tasks/a.html', function(status) {
    console.log("Page is opened");
});
```

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

```
Page is loaded
Page is opened
Page is loaded
```

On reload, the page is called again.

renderBase64 ()

This method takes the screen capture and gives the image as a string in base46. The renderbase64 method supports formats like PNG, JPEG and JPG. It does not support GIF as of now. You can use the **clipRect** property to define the portion for image capture.

Syntax

Its syntax is as follows:

```
wpage.renderBase64('PNG');
```

Example


```
var wpage = require('webpage').create();
wpage.open('http://localhost/taks/wopen2.html', function (status) {
    var base64 = wpage.renderBase64('PNG');
    console.log(base64);
    phantom.exit();
});
```

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

```
iVBORw0KGgoAAAANSUHEUgAAAZAAAAE3CAYAAACEpheaAAAACXBtWXMAAA7EAAAOxAGVKw4bAAAgAE1
E
QVR4nO3dv3OjSL8u8Ee33mA3uuXszGZnSnLd8jo6b4b+AmkS38R1TjQZCqXEmave3SpnTiC0am8w6UR
O
```

renderBuffer()

The renderBuffer method takes the capture of the webpage to an image buffer, which can be directly sent to the server. The formats supported are PNG, GIF and JPEG.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.renderBuffer(format, quality);
```

render()

Render helps to take the image buffer and save it as per the format specified. The formats supported are PDF, PNG, JPEG, BMP, PPM, GIF (support depends on the build of QT used)

Quality

It supports integers between 0 and 100. It is used mainly for JPEG and PNG formats. For JPEG, it is used in percentage. Level 0 will produce a very small and low quality file and 100 produces high quality file. The default value is 75. For PNG, it sets as a compression level with 0 having small file and 100 having higher one.

You can use **clipRect**, **viewportSize**, **paperSize** with render methods for rendering the image buffer in formats as required.

Syntax

Its syntax is as follows:

```
wpage.render(filename, {format: PDF|PNG|JPEG|BMP|PPM|GIF, quality: '100'});
```

Example: Image

Let us take an example to understand the use of **render()** method.

```
var wpage = require('webpage').create();
wpage.viewportSize = { width: 1920, height: 1080 };
wpage.open("http://www.google.com", function start(status) {
    wpage.render('image.jpeg', {format: 'jpeg', quality: '100'});
    phantom.exit();
});
```

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



Example: PDF

Let us consider another example.

```
var wpage = require('webpage').create();
var url = "https://jquery.com/download/";
var output = "display.pdf";

wpage.paperSize = {
    width: '600px',
    height: '1500px',
    margin: {'top': '50px',
```

```

    'left': '50px',
    'right': '50px'

  },
  orientation: 'portrait',
  header: {
    height: "1cm",
    contents: phantom.callback(function(pageNumber, nPages) {
      return "<h5>Header <b>" + pageNumber + " / " + nPages + "</b></h5>";
    })
  },
  footer: {
    height: "1cm",
    contents: phantom.callback(function(pageNumber, nPages) {
      return "<h5>Footer <b>" + pageNumber + " / " + nPages +
"</b></h5>";
    })
  }
}

wpage.open(url, function (status) {
  if (status !== 'success') {
    console.log('Page is not opening');
    phantom.exit();
  } else {
    wpage.render(output);
    phantom.exit();
  }
});

```

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

Saves as display.pdf with header and footer.

sendEvent ()

It is used to send an event to the webpage. They are not DOM events. Each of these events are sent to the webpage based on the user interaction.

The events supported by this method are mouse and keyboard events.

Mouse Events

```
SendEvent (mouseEventType [, mouseX, mouseY, button='left'])
```

MouseEventType: This is a type of event and it supports **mouseup**, **mousedown**, **mousemove**, **doubleclick** and **click**.

The **MouseX** and **MouseY** events are optional and takes the mouse position. The button parameter defines the button to push. It is on the left by default. For **mousemove**, there is no button pressed, so button is not considered.

Keyboard Events

```
SendEvent (keyboardEventType, keyOrKeys, [null, null, modifier])
```

KeyboardEventType: This is a type of event and supports **keyup**, **keypress** and **keydown**.

Keyorkeys: Second parameter is the key from `page.event.key` or a string. The third and fourth are not considered and need to pass NULL for it.

Modifier: It is a integer and has the following list:

- **0:** No modifier key is pressed.
- **0x02000000:** A Shift key on the keyboard is pressed.
- **0x04000000:** A Ctrl key on the keyboard is pressed.
- **0x08000000:** An Alt key on the keyboard is pressed.
- **0x10000000:** A Meta key on the keyboard is pressed.
- **0x20000000:** A keypad button is pressed.

Syntax

Its syntax is as follows:

```
sendEvent(mouseEventType[, mouseX, mouseY, button='left'])
```

Example

Let us take an example to understand the use of `sendEvent()` method.

```
var page = require('webpage').create();
page.onAlert = function(msg) {
    console.log(msg);
}
page.open('http://localhost/tasks/click.html', function(status) {
    var element = page.evaluate(function() {
```

```

    return document.querySelector('.mybutton');
  });
  page.sendEvent('click', element.offsetLeft, element.offsetTop, 'left');
  console.log('element is ' + element);
});

```

click.html

```

<html>
<body>

<form>
  <input type="button" class="mybutton" value="Click me" onclick="clickme()">
</form>

<p>welcome to phantomjs</p>

<script>
function clickme() {
    alert("Hello world!");
}
</script>

</body>
</html>

```

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

```

Hello world!
element is [object Object]

```

setContent ()

The setContent method will change the page content of the specified URL with the contents specified.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.setContent(content, pageurl);
```

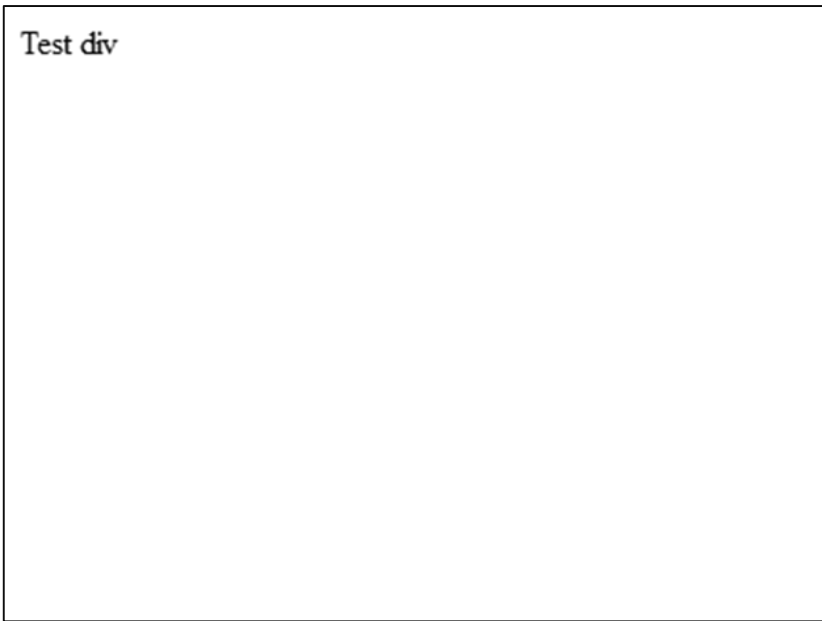
Example

The following example shows the use of **setContent()** method.

```
var wpage = require('webpage').create();
var Content = '<html><body><div>Test div</div></body></html>';
var pageurl = 'http://localhost/tasks/c.html';
wpage.setContent(Content, pageurl);
wpage.onLoadFinished = function(status) {
    wpage.render('newtest.png');
    phantom.exit();
};
```

Output – newtest.png

The above program generates the following output.



Test div

stop ()

The **stop()** method helps to stop the loading of a page.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.stop();
```

Example

```
var wpage = require('webpage').create();
wpage.onLoadStarted = function() {
    wpage.stop();
}
wpage.open('http://localhost/tasks/wopen2.html', function(status) {

    console.log(status);
    phantom.exit();
});
```

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

```
Fail
```

In the above example, we are using the **onLoadStarted** event to stop the page from loading. We get status as failed, since the page was stopped.

switchToFocusedFrame ()

The switchToFocusedFrame() method selects the frame that is in focus.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.switchToFocusedFrame();
```

Example

```
var page = require('webpage').create();
page.open('http://localhost/tasks/frames.html', function(status) {
    page.switchToFocusedFrame();
    console.log(page.focusedFrameName);
});
```

```
});
```

frames.html

```
<html>
<head><title>welcome to phantomjs</title></head>
<body name="content">
<script type="text/javascript">
window.name="page2";
console.log('welcome to cookie example');
document.cookie = "username=Roy; expires=Thu, 22 Dec 2017 12:00:00 UTC";
window.onload = function() {

    console.log("page is loaded");
}
</script>
<iframe src="http://localhost/tasks/a.html" width="800" height="800"
name="myframe" id="myframe">
</iframe>
<iframe src="http://localhost/tasks/content.html" width="800" height="800"
name="myframe1" id="myframe1">
</iframe>
<iframe src="http://localhost/tasks/click.html" width="800" height="800"
name="myframe2" id="myframe2">
</iframe>

<h1>Welcome to PhantomJS</h1>
</body>
</html>
```

The above program generates the following output.

```
page2
```


switchToFrame ()

Selects the frame with the name specified, which is also the child of the current frame.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.switchToFrame();
```

Example

```
var page = require('webpage').create();
page.open('http://localhost/tasks/frames.html', function(status) {
    page.switchToParentFrame();
    page.switchToFrame('myframe');
    console.log(page.frameName);
});
```

It will produce the following output:

```
Myframe
```

switchToMainFrame ()

The switchToMainFrame () method selects the main frame, i.e., the root window.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.switchToMainFrame();
```

Example

```
var page = require('webpage').create();
page.open('http://localhost/tasks/frames.html', function(status) {
    page.switchToMainFrame();
    console.log(page.frameName);
});
```

It will produce the following output:

```
page2
```

switchToParentFrame()

It takes the parent frame of the current child frame.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();  
wpage.switchToParentFrame();
```

Example

```
var page = require('webpage').create();  
page.open('http://localhost/tasks/frames.html', function(status) {  
    page.switchToParentFrame();  
    console.log(page.frameName);  
});
```

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

```
page2
```

uploadFile ()

This method is used to handle the file upload done using form in html. PhantomJS does not have a direct way to do so using forms, but the same can be achieved using the uploadFile method. It takes the html tag selector for the file location and the destination where it has to be copied.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();  
wpage.uploadFile('input[name=image]', 'path to copy file');
```

Example

The following example shows the use of **uploadFile()** method.

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

wpage.open("http://localhost/tasks/file.html", function(status) {
    console.log(status);
    wpage.uploadFile('input[name=fileToUpload]', 'output.png');

    wpage.render("result.png");
});
```

file.html

```
<html>
<head><title>Window 2</title></head>
<body>
<form action="upload.php" method="post" enctype="multipart/form-data"
id="form1">
    <input type="file" name="fileToUpload" id="fileToUpload">
    <input type="submit" value="Upload Image" name="submit">
</form>
</body>
</html>
```

The above program generates the following output.

Choose File

output.png

Upload Image

7. PhantomJS – Events/Callbacks

The callbacks available with the webpage module are described in the following table.

S No.	Callbacks & Description
1	onAlert() This callback is called when there is an alert on the page; the callback takes a string and does not return anything.
2	onCallback() OnCallback is used to pass values from webpage to webpage object and it is done using the window.callPhantom() method, which internally invokes the onCallback function.
3	onClosing() This is called when the window is closed either by using the page.close() method or the window.close() method.
4	onConfirm() This callback is called when there is a confirmed message called with ok /cancel from the page.
5	onConsoleMessage() This callback is used when there are console messages used in the webpage. The onConsoleMessage method takes 3 arguments.
6	onError() It is called when there is JavaScript error. The arguments for onError are msg and stack trace, which is an array.
7	onFilePicker() This is used for upload file the callback is called when user want to upload a file.
8	onInitialized() This callback is invoked when the page is called before loading.
9	onLoadFinished() This function is called when the page opened is fully loaded. It has one argument, which tells when loading was a success or a failure.

10	onLoadStarted() This callback is invoked when the page starts loading.
11	onNavigationRequested() This callback tells when the navigation event is taking place.
12	onPageCreated() This callback is invoked when a new child window is opened by the page.
13	onPrompt() This callback is called when a prompt is called by the web page. It takes 2 arguments, message and the answer. The return value is a string.
14	onResourceError() This callback is called when the webpage is not able to upload the file.
15	onResourceReceived() This callback is called when the resource requested by the page is received.
16	onResourceRequested() This function is invoked when page requests a resource.
17	onResourceTimeout() This callback is called when the requested resource times out, when settings.resourceTimeout is used.
18	onUrlChanged() This is called when the URL changes from the current one using navigation. It has one argument to the call back, which is a new URL targetUrl string.

Let us now discuss each of these events and callbacks in detail.

onAlert()

This callback is called when there is an alert on the page. The callback takes a string and does not return anything.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.onAlert = function(msg) {};
```

Example

The following example shows the use of **onAlert()** method.

```
var wpage = require('webpage').create();
wpage.onAlert = function(str) {
    console.log(str);
}
wpage.open('http://localhost/tasks/alert.html', function(status) {
    //wpage.stop();
    console.log(status);
    phantom.exit();
});
```

alert.html

```
<html>
<head></head>
<body>
<script type="text/javascript">
alert("Welcome to phantomjs");
</script>
<h1>This is a test page</h1>
</body>
</html>
```

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

```
Welcome to phantomjs
Success
```

onCallback ()

The **onCallback** method is used to pass values from a webpage to a webpage object and it is done using the `window.callPhantom()` method. This method internally invokes the `onCallback` function.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.onCallback = function(data) {}
```

Example

The following example shows the use of **onCallback()** method.

```
var wpage = require('webpage').create();
wpage.onCallback = function(str) {
    console.log(str + " phantomJs");
}
wpage.open('http://localhost/tasks/callback.html', function(status) {
    console.log(status);
    phantom.exit();
});
```

callback.html

```
<html>
<head></head>
<body>
<script type="text/javascript">
    var a = window.callPhantom("Welcome to ");
</script>
</body>
</html>
```

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

```
Welcome to phantomJs
Success
```

onClosing ()

This method is called when the window is closed either by using **page.close()** function or the **window.close()** function.

Syntax

Its syntax is as follows:

```
var wpage = require('webpage').create();
wpage.onClosing = function(str) {}
```


Example

```
var wpage = require('webpage').create();
wpage.onClosing = function(str) {
    console.log("page is closed");
}
wpage.open('http://localhost/tasks/closing.html', function(status) {
    console.log(status);
    wpage.close();
});
```

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

```
success
page is closed
```

onConfirm ()

This callback is called when there is a confirmed message called with **ok /cancel** from the page. Its return value to the callback is true or false, which is true for **ok** and false for **cancel** clicked on the confirm box.

Syntax

Its syntax is as follows:

```
wpage.onConfirm = function(msg) {};
```

Example

The following example shows the use of **onConfirm()** method.

```
var wpage = require('webpage').create();
wpage.onConfirm = function(msg) {
    console.log(msg);
}
wpage.open('http://localhost/tasks/confirm.html', function(status) {
    console.log(status);
    phantom.exit();
});
```

confirm.html

```
<html>
<body>
<script>
window.confirmValue = confirm("Press a button!");
</script>
</body>
</html>
```

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

```
Press a button!
Success
```

onConsoleMessage ()

This callback is used when there are console messages used in the webpage. This callback takes the following three arguments.

- Message
- Line no
- Source identifier

Syntax

Its syntax is as follows:

```
page.onConsoleMessage = function(msg, lineNum, sourceId) {};
```

Example

```
var wpage = require('webpage').create();
wpage.onConsoleMessage = function(msg) {
    console.log('CONSOLE Message: ' + msg );
};
wpage.open('http://localhost/tasks/test.html', function(status) {
    var script1 = "function(){ console.log('hello world');}";
    var value = wpage.evaluate(script1);
    phantom.exit();
});
```

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

```
CONSOLE Message: hello world
```

onError()

The OnError() method is called when there is a JavaScript error. The arguments for the onError method are **msg** and **stack trace**, which is an array.

Syntax

Its syntax is as follows:

```
page.onError = function(msg, trace) {}
```

Example

The following code shows the use of **onError()** method.

```
var wpage = require('webpage').create();
wpage.onError = function(msg, trace) {
    console.log('CONSOLE Message: ' + msg );
    console.log(JSON.stringify(trace));
};
wpage.open('http://localhost/tasks/test.html', function(status) {
    phantom.exit();
});
```

test.html

```
<html>
<head>
<title>Welcome to phantomjs</title>
</head>
<body>
<script type="text/javascript">
window.onload = function(){
    console.log(page);
}
</script>
<h1>This is a test page</h1>
```

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

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

```
CONSOLE Message: ReferenceError: Can't find variable: page
[{"file":"http://localhost/tasks/test.html","line":8,"function":"onload"}]
```

onFilePicker ()

This onFilePicker () callback is used for uploading the file. This callback is called when the user wants to upload a file.

Syntax

Its syntax is as follows:

```
page.onFilePicker = function(oldFile) { }
```

Example

```
var wpage = require('webpage').create();
wpage.onFilePicker = function(oldFile) {
    console.log('Entering file picker callback');
    console.log(oldFile);
}
wpage.open("http://localhost/tasks/file.html", function(status) {
    console.log(status);
    wpage.evaluate(function() {
        var fileUp = document.querySelector("#fileToUpload");
        var ev = document.createEvent("MouseEvents");
        ev.initEvent("click", true, true);
        fileUp.dispatchEvent(ev);
    });
});
```

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

```
Success
Entering file picker callback
```

onInitialized ()

This callback is invoked when the page is called and before it is loaded.

Syntax

Its syntax is as follows:

```
page.onInitialized = function() {}
```

Example

```
var wpage = require('webpage').create();
wpage.onInitialized = function() {
    console.log('File is initialized');
}
wpage.open("http://localhost/tasks/file.html", function(status) {
    console.log(status);
});
```

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

```
Success
File is initialized
```

onLoadFinished ()

This function is called when the page opened is fully loaded. It has a one argument, which tells whether the loading was a success or a failure.

Syntax

Its syntax is as follows:

```
wpage.onLoadFinished = function(status) {}
```

Example

```
var wpage = require('webpage').create();
wpage.onLoadFinished = function(str) {
    console.log(str);
    console.log('File is Loaded');
}
```

```
wpage.open("http://localhost/tasks/file.html", function(status) {
});
```

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

```
success
File is Loaded
```

onLoadStarted ()

This callback is called when the page starts loading.

Syntax

Its syntax is as follows:

```
wpage.onLoadStarted = function() {}
```

Example

```
var wpage = require('webpage').create();
wpage.onLoadStarted = function() {
    console.log('File has started Loading');
}
wpage.open("http://localhost/tasks/file.html", function(status) {
    console.log(status);
});
```

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

```
File has started Loading
Success
```

onNavigationRequested ()

This callback tells when the navigation event is taking place. It takes the following four arguments:

- **URL:** The target URL of the navigation event.
- **Type:** Values for type are undefined, Linkclicked, FormSubmitted, BackorForward, Reload, FormReSubmitted, Other.
- **willNavigate:** It is true, if navigation will take place, false if it is locked.

- **Main:** It is true, if it comes from main window, false if it comes from iframe or any other sub frame.

Syntax

Its syntax is as follows:

```
wpage.onNavigationRequested = function(url, type, willNavigate, main) {}
```

Example

```
var wpage = require('webpage').create();
wpage.onNavigationRequested = function(url, type, willNavigate, main) {
    console.log('Trying to navigate to: ' + url);
    console.log('Caused by: ' + type);
    console.log('Will actually navigate: ' + willNavigate);
    console.log('Sent from the page\'s main frame: ' + main);
}
wpage.open('http://localhost/tasks/wopen2.html', function(status) {
    console.log(status);
    if (status == success) {
        console.log(wpage.content);
        wpage.reload();
    }
});
```

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

```
Trying to navigate to: http://localhost/tasks/wopen2.html
Caused by: Other
Will actually navigate: true
Sent from the page's main frame: true
Success
```

We are invoking the navigate callback on the reload of the page.

onPageCreated ()

This callback is invoked when a new child window is opened by the page.

Syntax

Its syntax is as follows:

```
wpage.onPageCreated = function(newPage) {}
```

Example

The following example shows the use of **onPageCreated()** method.

```
var wpage = require('webpage').create();
wpage.onPageCreated = function(newpage) {
    console.log("Entering in onPageCreated callback");
    console.log(newpage.url);
}
wpage.open('http://localhost/tasks/newwindow.html', function(status) {
    console.log(status);
});
```

newwindow.html

```
<html>
<head>
<title>Welcome to phantomjs</title>
</head>
<body>
<script type="text/javascript">
window.onload = function(){
    window.open("test.html", "window1");
}
</script>
<h1>This is a test page</h1>
</body>
</html>
```


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

```
Entering in onPageCreated callback
WebPage(name = "WebPage")
Success
```

onPrompt()

This callback is called when a prompt is called by the webpage. It takes two arguments, **message** and the **answer**. The return value is a string.

Syntax

Its syntax is as follows:

```
wpage.onPrompt = function(msg, defaultVal) {}
```

Example

The following code shows the use of **onPrompt()** method.

```
var wpage = require('webpage').create();
wpage.onPrompt = function(msg, answer) {
    console.log("Entering in onPrompt callback");
    console.log(msg);
    return answer;
}
wpage.open('http://localhost/tasks/prompt.html', function(status) {
    console.log(status);
});
```

prompt.html

```
<html>
<head>
<title>Welcome to phantomjs</title>
</head>
<body>
<script type="text/javascript">
window.onload = function() {
    prompt("Is the page loaded", "");
```

```

}
</script>
<h1>This is a test page</h1>
</body>
</html>

```

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

```

Entering in onPrompt callback
Is the page loaded
Success

```

onResourceError ()

This callback is called when the webpage is not able to upload the file. The arguments to it is the error object.

The error object contains:

- **Id:** The number of the request.
- **URL:** The URL called.
- **ErrorCode:** error code.
- **ErrorString:** error details.

Syntax

Its syntax is as follows:

```

wpage.onResourceError = function(resourceError) {}

```

Example

The following example shows the use of **onResourceError()** method.

```

var wpage = require('webpage').create();
wpage.onResourceError = function(error) {
    console.log(JSON.stringify(error));
}
wpage.open('http://localhost/tasks/prompt1.html', function(status) {
});

```

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

```
{ "errorCode": 203, "errorString": "Error downloading http://localhost/tasks/prompt1.html - server replied: Not Found", "id": 1, "status": 404, "statusText": "Not Found", "url": "http://localhost/tasks/prompt1.html" }
```

onResourceReceived ()

This callback is called when the resource requested by the page is received. It contains response as the argument.

The response object has the following details:

- **Id:** number of the requested resource.
- **URL:** The requested URL.
- **Time:** The date object containing date of the response.
- **Headers:** http headers.
- **BodySize:** The size of the received content decompressed.
- **ContentType:** The content type if specified.
- **RedirectURL:** If there is redirection then the redirected URL.
- **Stage:** The values are – start and end.
- **Status:** The http code status i.e. status 200.
- **StatusText:** The http status text for code 200, it is OK.

Syntax

Its syntax is as follows:

```
page.onResourceReceived = function(response) {}
```

Example

The following code shows the use of **onResourceReceived()** method.

```
var wpage = require('webpage').create();
wpage.onResourceReceived = function(response) {
    console.log(JSON.stringify(response));
}
wpage.open('http://localhost/tasks/prompt.html', function(status) {
});
```

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

```
{
  "body": "",
  "bodySize": 231,
  "contentType": "text/html",
  "headers": [
    { "name": "Date", "value": "Sun, 07 May 2017 12:59:17 GMT" },
    { "name": "Server", "value": "Apache/2.4.17 (Win32) OpenSSL/1.0.2d PHP/5.6.23" },
    { "name": "Last-Modified", "value": "Sun, 07 May 2017 12:48:14 GMT" },
    { "name": "ETag", "value": "\"e7-54eee893517e5\"" },
    { "name": "Accept-Ranges", "value": "bytes" },
    { "name": "Content-Length", "value": "231" },
    { "name": "Keep-Alive", "value": "timeout=5, max=100" },
    { "name": "Connection", "value": "Keep-Alive" },
    { "name": "Content-Type", "value": "text/html" }
  ],
  "id": 1,
  "redirectURL": null,
  "stage": "start",
  "status": 200,
  "statusText": "OK",
  "time": "2017-05-07T12:59:17.440Z",
  "url": "http://localhost/tasks/prompt.html"
}

{
  "contentType": "text/html",
  "headers": [
    { "name": "Date", "value": "Sun, 07 May 2017 12:59:17 GMT" },
    { "name": "Server", "value": "Apache/2.4.17 (Win32) OpenSSL/1.0.2d PHP/5.6.23" },
    { "name": "Last-Modified", "value": "Sun, 07 May 2017 12:48:14 GMT" },
    { "name": "ETag", "value": "\"e7-54eee893517e5\"" },
    { "name": "Accept-Ranges", "value": "bytes" },
    { "name": "Content-Length", "value": "231" },
    { "name": "Keep-Alive", "value": "timeout=5, max=100" },
    { "name": "Connection", "value": "Keep-Alive" },
    { "name": "Content-Type", "value": "text/html" }
  ],
  "id": 1,
  "redirectURL": null,
  "stage": "end",
  "status": 200,
  "statusText": "OK",
  "time": "2017-05-07T12:59:17.486Z",
  "url": "http://localhost/tasks/prompt.html"
}
```

onResourceRequested ()

This is invoked when the page requests a resource. It has two arguments **requestData** and **networkRequest**.

The RequestData object has the following details:

- **Id:** The number of the requested resource.
- **Method:** The http method.
- **URL:** The URL of the requested resource.
- **Time:** the date object containing the date of the request.
- **Headers:** the list of the http headers.

The NetworkRequest object has the following details:

- **Abort ():** It aborts the current network request. Aborting the current network request will invoke the onResourceError callback function.
- **ChangeUrl (newurl):** The requested URL can be changed to a new file using this function.
- **SetHeader:** It has the key and the value.

Syntax

Its syntax is as follows:

```
page.onResourceRequested = function(requestData, networkRequest) {}
```

Example

```
var wpage = require('webpage').create();
wpage.onResourceRequested = function(requestdata , networkdata) {
    console.log("Data from requestdata:");
    console.log(JSON.stringify(requestdata));
    console.log("Data from networkdata");
    console.log(JSON.stringify(networkdata));
}
wpage.open('http://localhost/tasks/request.html', function(status) {
});
```

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

```
Data from requestdata:
{"headers":[{"name":"Accept","value":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"}, {"name":"User-Agent","value":"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/538.1 (KHTML, like Gecko) PhantomJS/2.1.1 Safari/538.1"}],
"id":1,"method":"GET","time":"2017-05-07T13:25:36.454Z","url":"http://localhost/tasks/request.html"}
Data from networkdata
{"objectName":""}
```

onResourceTimeout ()

This callback is called when the requested resource timesout. That is when the settings.resourceTimeout callback is used.

It contains one argument i.e. **request** that have the following details:

- **id:** The number of the requested resource
- **Method:** The http method
- **URL:** The URL of the requested resource
- **Time:** Date object containing the date of the request
- **Headers:** List of the http headers
- **ErrorCode:** The error code of the error
- **ErrorString:** The text message of the error

Syntax

Its syntax is as follows:

```
page.onResourceTimeout = function(request) {}
```

Example

```
var wpage = require('webpage').create();
wpage.onResourceTimeout = function(request) {
    console.log("Data from request:");
    console.log(JSON.stringify(request));
}
wpage.settings.resourceTimeout = '3';
wpage.open('http://localhost/tasks/request.html', function(status) {
});
```

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

```
Data from request:
{"errorCode":408,"errorString":"Network timeout on
resource.", "headers":[{"name":"Accept","value":"text/html,application/xhtml+xml
,application/xml;q=0.9,*/*;q=0.8"}, {"name":"User-Agent","value":"Mozilla/5.0
(Windows NT 6.2; WOW64) AppleWebKit
it/538.1 (KHTML, like Gecko) PhantomJS/2.1.1
Safari/538.1"}], "id":1, "method":"GET", "time":"2017-05-
07T13:32:12.545Z", "url":"http://localhost/tasks/request.html"
}
```

onUrlChanged ()

This is called when the URL changes from the current one using navigation. It has one argument to the callback, which is the new URL **targetUrl** string.

Syntax

Its syntax is as follows:

```
page.onUrlChanged = function(targetUrl) {}
```

Example

```
var wpage = require('webpage').create();
wpage.onUrlChanged = function(targeturl) {
```

```
    console.log("Entered onUrlChanged callback:");  
    console.log(targeturl);  
}  
wpage.settings.resourceTimeout = '3';  
wpage.open('http://localhost/tasks/request.html', function(status) {  
    var Content = '<html><body><div>Test div</div></body></html>';  
    var pageurl = 'http://localhost/tasks/c.html';  
    wpage.setContent(Content, pageurl);  
});
```

The above program will generate the following **output**:

```
Entered onUrlChanged callback:  
http://localhost/tasks/c.html
```

8. PhantomJS – Child Process Module

The Phantomjs Child process module helps to interact with the sub-processes and talk to them using **stdin /stdout/stderr**. The child processes can be used for tasks like **printing, sending mail** or to **invoke programs** written in another language. To create a child process module, you need references.

For example:

```
var process = require("child_process");
```

Spawn Method

With the spawn child process, you can subscribe to its **stdout** and **stderr** streams to get data real-time.

Syntax

Its syntax is as follows:

```
var spawn = require('child_process').spawn;
```

Example

Let us look at an example of the spawn method.

```
var process = require("child_process")
var spawn = process.spawn

var child = spawn("cmd", ['/c', 'dir']);

child.stdout.on("data", function (data) {
  console.log("spawnSTDOUT---VALUE:", JSON.stringify(data))
})

child.stderr.on("data", function (data) {
  console.log("spawnSTDERR:", JSON.stringify(data))
})

child.on("exit", function (code) {
  console.log("spawnEXIT:", code)
})
```


Output

The above program generates the following output.

```
spawnSTDOUT---VALUE: " Volume in drive C is OS\r\n"
spawnSTDOUT---VALUE: " Volume Serial Number is 7682-9C1B\r\n\r\n Directory of C:
\\phantomjs\\bin\r\n\r\n"
spawnSTDOUT---VALUE: "20-05-2017  10:01    <DIR>                .\r\n20-05-2017  10:01
    <DIR>                ..\r\n13-05-2017  20:48                12 a,txt.txt\r\n07-05-
2017  08:51                63 a.js\r\n06-05-2017  16:32                120,232 a.pdf\
r\n13-05-2017  20:49
spawnEXIT: 0
```

File System Module

9. PhantomJS – Properties

The File System Module has many APIs to deal with files and directories. You can create/write and delete a file/directory. To start using the file system module you must require a reference to the **fs module**.

```
var fs = require('fs');
```

There are two properties available for file system module: **Separator** and **Working Directory**. Let us understand them in detail.

Separator

It tells you the separator used for the file paths.

- For windows: \
- For Linux: /

Syntax

Its syntax is as follows:

```
fs.seperator
```

Example

```
var fs = require('fs');  
console.log(fs.seperator);  
phantom.exit();
```

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

```
Undefined
```

workingDirectory

The working directory is the directory in which PhantomJS executes.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.workingDirectory;
```

Example

```
var fs = require('fs');  
console.log(fs.workingDirectory);  
phantom.exit();
```

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

```
C:/phantomjs/bin
```

10. PhantomJS – Methods

The following table has all the methods and their descriptions, which are available on the File System module.

S No.	Methods & Description
1	absolute This method gives the absolute path where PhantomJS runs.
2	changeWorkingDirectory This allows you to change the working directory and returns true, if it succeeds otherwise returns false.
3	copyTree copyTree will copy a directory from one path to another. The first parameter is a source folder and the second parameter is a destination folder.
4	copy This method helps to copy file from one location to another. It takes two parameters. The first parameter is the source file and the second parameter is the file path, where it has to be copied. It will throw an error, if the source or destination file does not exist.
5	exists It checks whether the given file path exists in the system. It returns true, if it is present, otherwise false.
6	isAbsolute This method will return true, if the file path is absolute and false, if relative.
7	isDirectory isDirectory tells if the given file is a directory or not. It returns true, if directory otherwise false. It gives false in case if the given path does not exist.
8	isExecutable This method will tell if the given file is executable or not. It returns true, if executable, otherwise false.
9	isFile This gives details whether the filepath given is a file or not. It returns true, if it is a file and false, if it is not.
10	isLink This will give you true, if the file path is a symlink, otherwise false.
11	sReadable It checks if the given file is readable or not. It gives true, if yes and false, if not.

12	sWritable It tells whether if a given file is writable. Returns true, if yes, otherwise false.
13	lastModified Gives the details of the last modification done to the file. It tells the date and time of when the file was last modified.
14	list It gives all the files present in the directory.
15	makeDirectory Creates a new directory.
16	makeTree makeTree creates all necessary folders to be able to form final directories. This will return true, if the creation was successful, otherwise false. If the directory already exists, it will return true.
17	move It will move the file from one path to another.
18	open It is used to open up the file.
19	readLink This will return the absolute path of a file or a folder pointed by a symlink (or shortcut on Windows). If the path is not a symlink or shortcut, it will return an empty string.
20	read This will read the given file.
21	removeDirectory This will remove the given directory.
22	removeTree It will delete all the files and folders from the given folder and finally delete the folder itself. If there is any error while doing this process, it will throw an error – 'Unable to remove directory tree PATH' and hang execution.
23	remove It removes the given file.
24	size It gives the size of the file.
25	touch It creates a given file.
26	write Writes to a given file.

absolute

This method gives the absolute path where PhantomJS runs.

Syntax

Its syntax is as follows:

```
fs.absolute();
```

Example

Let us look at an example of the **absolute** method.

```
var fs = require('fs');  
var cwd = fs.absolute(".");  
console.log(fs);
```

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

```
C:/phantomjs/bin
```

changeWorkingDirectory

It allows you to change the working directory and returns true, if successful, otherwise returns false.

Syntax

Its syntax is as follows:

```
fs.changeWorkingDirectory(filepath);
```

Example

Let us look at an example of the changeWorkingDirectory method.

```
var fs = require('fs');  
console.log(fs.workingDirectory); // prints the location where phantomjs is running  
fs.changeWorkingDirectory("C:\\");  
console.log(fs.workingDirectory); // prints C:/
```

The above program generates the following output.

```
C:/phantomjs/bin  
C:/
```

copyTree

The copyTree method will copy a directory from one path to another. The first parameter is the **source** folder and second parameter is the **destination** folder. If the destination does not exist, it will be created and every file and folder from the source folder will be copied to the destination folder.

Folders will be recursively copied, if any of the file or folder fails while copying, it will throw an error – "Unable to copy directory tree SOURCE at DESTINATION" and execution will hang.

Syntax

Its syntax is as follows:

```
copyTree(source,destination);
```

Example

The following example shows the use of **copyTree** method.

```
var fs = require('fs');
var system = require('system');
var path1 = system.args[1];
var path2 = system.args[2];
console.log("Checking to see if source is a file:" + fs.isDirectory(path1));
console.log("Checking to see if destination is a file:" +
fs.isDirectory(path2));
console.log("copying tree directory from source to destination");
fs.copyTree(path1, path2);
console.log("Checking to see if destination is a file:" +
fs.isDirectory(path2));
```

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

Command: phantomjs copytree.js newdirectory/a/b/c/file.txt destfolder

```
Checking to see if source is a file:true
Checking to see if destination is a file:false
copying tree directory from source to destination
Checking to see if destination is a file:true
```


copy

This method helps to copy a file from one location to another. It takes two parameters. The first parameter is the **source file** and the second parameter is the **file path**, where it has to be copied. It will throw an error if the source or destination file does not exist.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.copy(sourcefile,destinationfile);
```

Example

```
var fs = require('fs');  
var system = require('system');  
var path1 = system.args[1];  
var path2 = system.args[2];  
console.log("Checking to see if source is a file:" + fs.isFile(path1));  
console.log("Checking to see if destination is a file:" + fs.isFile(path2));  
console.log("copying file from source to destination");  
fs.copy(path1,path2);  
console.log("Checking to see if destination is a file:" + fs.isFile(path2));  
console.log("contents from file are:");  
console.log(fs.read(path2));
```

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

```
var fs = require('fs');  
var system = require('system');  
var path1 = system.args[1];  
var path2 = system.args[2];  
console.log("Checking to see if source is a file:" + fs.isFile(path1));  
console.log("Checking to see if destination is a file:" + fs.isFile(path2));  
console.log("copying file from source to destination");  
fs.copy(path1,path2);  
console.log("Checking to see if destination is a file:" + fs.isFile(path2));  
console.log("contents from file are:");  
console.log(fs.read(path2));
```

exists

The **exists** method checks whether a given filepath exists in the system or not. It returns true, if it is present, otherwise false.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.exists(filepath);
```

Example

The following example shows the use of **exists()** method.

Command: phantomjs exists.js C:\test\

```
var fs = require('fs');  
var system = require('system');  
console.log(fs.exists(system.args[1]));  
phantom.exit();
```

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

```
True
```

isAbsolute

This method returns **true** if the filepath is absolute. If the filepath is relative, it returns **false**.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.isAbsolute(path);
```

Example

The following example shows the use of **isAbsolute()** method.

```
var fs = require('fs');  
var path = 'C:\phantomjs\bin\a.txt';
```

```
if (fs.isAbsolute(path)) {  
    console.log(' Absolute path.');
```

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

```
Absolute path.
```

isDirectory

The **isDirectory** function checks if a given file is a directory or not. It returns **true**, if the file is a directory, otherwise **false**. It returns **false** in case the given path does not exist.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.isDirectory(filepath);
```

Example

The following example shows how you can use the **isDirectory** method.

Command: phantomjs isdirectory.js C:\test\

```
var fs = require('fs');  
var system = require('system');  
console.log(fs.isDirectory(system.args[1]));  
phantom.exit();
```

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

```
True
```

isExecutable

This method checks if a given file is executable or not. It returns true, if executable; otherwise, false.

Syntax

Its syntax is as follows:

```
var fs=require('fs');  
fs.isExecutable(filename);
```

Example

The following example shows the use of **isExecutable** method.

Command: Phantomjs isexecutable.js touch.js

```
var fs=require('fs');  
var system = require('system');  
var path = system.args[1];  
if (fs.isExecutable(path)) {  
    console.log(path+" is executable.");  
} else{  
    console.log(path+" is NOT executable.");  
}  
phantom.exit();
```

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

```
touch.js is NOT executable.
```

isFile

This method checks whether the supplied filepath is a file or not. It returns **true** if the filepath is a file; **false** otherwise.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.isFile(filepath);
```

Example

The following example shows the use of **isFile** method.

Command: phantomjs isfile.js isfile.js

```
var fs = require('fs');  
var system = require('system');  
console.log(fs.isFile(system.args[1]));  
phantom.exit();
```

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

```
True
```

isLink

This method returns **true**, if the given filepath is a symlink; **false** otherwise.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.isLink(filepath);
```

Example

The following code shows how to use the **isLink** method.

Command: phantomjs islink.js phantomjs\bin\touch

```
var fs = require('fs');  
var system = require('system');  
var path = system.args[1];  
if(fs.isLink(path))  
    console.log('"' + path + '" is a link.');
```

```
else  
    console.log('"' + path + '" is an absolute path');
```

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

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

```
"phantomjs\bin\touch" is an absolute path
```

isReadable

It checks if a given file is readable or not. It returns **true** if yes, else **false**.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.isReadable(filename);
```

Example

The following example shows the use of **isReadable** method.

Command: phantomjs isfile.js isfile.js

```
var fs = require('fs');  
var system = require('system');  
console.log(fs.isReadable(system.args[1]));  
phantom.exit();
```

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

```
True
```

isWritable

It checks if a given file is writable. Returns **true** if yes, otherwise **false**.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.isWritable(filename);
```

Example

The following example shows the use of **isWritable** method.

Command: Phantomjs isfile.js isfile.js

```
var fs = require('fs');  
var system = require('system');  
console.log(fs.isWritable(system.args[1]));
```

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

The above program generates the following output.

```
True
```

lastModified

This method returns the details of the last modification done to a given file. It returns the date and time when the file was last modified.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.lastModified(filename);
```

Example

```
var fs = require('fs');  
var system = require('system');  
console.log(fs.lastModified(system.args[1]));  
phantom.exit();
```

The above program generates the following output.

```
Sat May 13 2017 15:18:59 GMT+0530 (India Standard Time)
```

list

The **list** method returns all the files present in a directory.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.list(path);
```

Example

The following example shows the use of **list** method.

Command: phantomjs list.js C:\phantomjs\

```
var fs = require('fs');
var system = require('system');
var path = system.args[1];
if (fs.isDirectory(path)) {
    console.log(fs.list(path));
}
phantom.exit();
```

The above program generates the following output.

```
.,.,.,bin,ChangeLog,examples,LICENSE.BSD,README.md,third-party.txt
```

The output shows all the files present in the PhantomJS directory.

makeDirectory

This method creates a new directory.

Syntax

Its syntax is as follows:

```
var fs = require('fs');
fs.makeDirectory(path);
```

Example

Command: phantomjs makedirectory.js newfiles

```
var fs = require('fs');
var system = require('system');
var path = system.args[1];
var md = fs.makeDirectory(path);
console.log(fs.isDirectory(path));
phantom.exit();
```

The above program generates the following output.

```
True
```


makeTree

The **makeTree** method creates all the necessary folders to be able to form the final directories. This will return true, if the creation was successful, otherwise false. If the directory already exists, it will return true.

Syntax

Its syntax is as follows:

```
fs.makeTree(path);
```

Example

The following example shows how you can use the makeTree method:

```
var fs = require('fs');
var system = require('system');
var path = system.args[1];
fs.makeTree(path);
console.log("Checking to see if the maketree has converted into directory : "
+fs.isDirectory(path));
console.log("Listing the contents from the path given");
var a = path.split("/");
var list = fs.list(a[0]+"/"+a[1]+"/"+a[2]+"/"+a[3]);
console.log(JSON.stringify(list));
phantom.exit();
```

The above program generates the following output.

```
Checking to see if the maketree has converted into directory : true
Listing the contents from the path given
[".", "..", "file.txt"]
```

move

This method moves a specified file from one path to another. For example, "**move (source, destination)**". Here, the first parameter is the source file and the second parameter is the destination path with the file name. If the source file cannot be found, then it will throw an "Unable to copy file SOURCE at DESTINATION" error and hang execution.

If the destination cannot be created, then it will throw an "Unable to copy file SOURCE at DESTINATION" error and hang execution. It will not overwrite the existing files. If the

source file cannot be deleted, then it will throw an 'Unable to delete file SOURCE' error and hang.

Syntax

Its syntax is as follows:

```
fs.move(sourcefilepath, destinationfilepath)
```

Example

Let us take an example to understand how the **move** method works.

```
var fs = require('fs');
var system = require('system');
var sourcefile = system.args[1];
var destfile = system.args[2];
console.log("Checking if sourcefile is a file : " +fs.isFile(sourcefile));
console.log("Checking if destfile is a file : " +fs.isFile(destfile));
console.log("moving the files");
fs.move("openmode.txt", "newfiles/move.txt");
console.log("Content from move.txt: ");
console.log(fs.read("newfiles/move.txt"));
console.log("Checking if sourcefile is a file : " +fs.isFile(sourcefile));
```

The above program generates the following output.

```
Checking if sourcefile is a file : true
Checking if destfile is a file : false
moving the files
Content from move.txt:
This is used for testing.
Checking if sourcefile is a file : false
```

open

The Open method takes two parameters, which are – **file path** and the **object** parameter.

The object parameter can contain:

- **Mode:** Open Mode. A string made of 'r', 'w', 'a/+', 'b' characters.
- **Charset:** An IANA, case insensitive, charset name.

Stream objects are returned from the **fs.open** method. The stream has to be closed once open. When errors occur during a call, it will throw an 'Unable to open file PATH' error and hang execution.

Syntax

Its syntax is as follows:

```
fs.open('filepath', 'r or w or a/+ or b');
```

Example

Let us take an example to understand how the **open** method works.

```
var fs = require('fs');  
var file = fs.open('openmode.txt', 'r');  
console.log(file);  
file.close();
```

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

```
File(name = "")
```

readLink

This method returns the absolute path of a file or folder pointed by a symlink (or shortcut on Windows). If the path is not a symlink or shortcut, it will return an empty string.

Syntax

Its syntax is as follows:

```
fs.readLink(path);
```

Example

The following example shows how the **readLink** method works.

```
var fs = require('fs');  
var path = 'C:\\phantomjs\\bin\\readlink.txt';  
  
if (fs.isLink(path)) {  
    var absolute = fs.readLink(path);  
    console.log('The absolute path of "'+path+'" is "'+absolute+'");  
}
```

```
else
    console.log(''+path+'' is an absolute path');

phantom.exit();
```

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

```
readlink.txt
```

read

This method reads the data present in a specified file.

Syntax

Its syntax is as follows:

```
var fs = require('fs');
fs.read(path);
```

Example

Let us take a look at an example to understand how the **read** method works.

Command: phantomjs read.js test.txt

```
var fs = require('fs');
var system = require('system');
var path = system.args[1];
console.log(fs.read(path));
phantom.exit();
```

Test.txt file

```
I am Testing Phantomjs
```

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

```
I am Testing Phantomjs
```

removeDirectory

This method helps to remove a given directory.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.removeDirectory(path);
```

Example

The following example shows how the **removeDirectory** method works.

Command: phantomjs removedirectory.js test

```
var fs = require('fs');  
var system = require('system');  
var path = system.args[1];  
console.log("Creating new directory " + path);  
var md = fs.makeDirectory(path);  
console.log("Directory present : "+fs.isDirectory(path));  
console.log("Removing directory "+path);  
var rd = fs.removeDirectory(path);  
console.log("Directory present "+fs.isDirectory(path));  
phantom.exit();
```

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

```
Creating new directory test  
Directory present : true  
Removing directory test  
Directory present false
```

removeTree

The removeTree method is used to delete all the files and folders from a given folder and finally delete the folder itself. If there is any error while doing this process, it will throw an error – "Unable to remove directory tree PATH" and hang execution.

Syntax

Its syntax is as follows:

```
fs.removeTree(foldertodelete)
```

Example

The following example shows how the **removeTree** method works.

```
var fs = require('fs');
var system = require('system');
var path = system.args[1];
console.log("Directory present : "+fs.isDirectory(path));
var a = fs.list(path);
console.log("Listing the contents from the directory : " + JSON.stringify(a));
console.log("Removing directory "+path);
var rd = fs.removeTree(path);
console.log("Directory present "+fs.isDirectory(path));
phantom.exit();
```

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

```
Directory present : true
Listing the contents from the directory : [".","..","examples","newfiles"]
Removing directory removetree
Directory present false
```

remove

This method is used to remove a specified file.

Syntax

Its syntax is as follows:

```
var fs = require('fs');
fs.remove(path);
```

Example

The following example shows how the **remove** method works.

Command: phantomjs remove.js test.txt

```
var fs = require('fs');
var system = require('system');
var path = system.args[1];
var md = fs.isFile(path);
console.log("File is present : "+fs.isFile(path));
console.log("Removing file "+path);
var rd = fs.remove(path);
console.log("File is present "+fs.isFile(path));
phantom.exit();
```

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

```
File is present: true
Removing file test.txt
File is present false
```

size

This method returns the size of the file.

Syntax

Its syntax is as follows:

```
var fs = require('fs');
fs.size(path);
```

Example

```
var fs = require('fs');
var system = require('system');
var path = system.args[1];
var md = fs.size(path);
console.log("Size of file is : "+md);
phantom.exit();
```

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

```
Size of file is : 625
```

touch

This method creates an empty file.

Syntax

Its syntax is as follows:

```
var fs = require('fs');  
fs.touch(path);
```

Example

The following example shows how the **touch** method works.

Command: phantomjs touch.js hello.txt

```
var fs = require('fs');  
var system = require('system');  
var path = system.args[1];  
var md = fs.touch(path);  
console.log("file is present : "+fs.isFile(path));  
phantom.exit();
```

It will generate the following output.

```
file is present : true
```

write

This method takes three parameters: **Source**, **Content**, and **String/Object**.

- **Source:** It is the file, where the content has to be written.
- **Content:** This parameter is the content that needs to be written to the file.
- **String / Object:** This parameter has the mode and charset.
 - **Mode:** Open mode. A string made of 'r', 'w', 'a/+', 'b' characters.
 - **Charset:** An IANA, case insensitive, charset name.

Syntax

Its syntax is as follows:

```
fs.write(path, content, 'w');
```

Example

The following example shows how the **write** method works.

Command: phantomjs write.js writemode.txt

```
var fs = require('fs');
var system = require('system');
var path = system.args[1];
var md = fs.touch(path);
console.log("file is present : "+fs.isFile(path));
var n = fs.write(path, 'Hello world', 'w');
console.log("Content present in " +path + " are :");
console.log(fs.read(path));
phantom.exit();
```

The above program generates the following output.

```
file is present : true
Content present in writemode.txt are :
Hello world
```

System Module

11. PhantomJS – Properties

In this chapter, we will discuss regarding the various System Module Properties of PhantomJS.

args

The system module of PhantomJS has different properties, which help to pass arguments, get to know the file that is executed, etc.

The args property basically returns the arguments passed at the command line. You can also pass the page-URL that you want to read. If you want to store the screen-capture pdf-file or upload file, the details can be passed to the command line and can be read using the **args** property.

Let us find the length and the way to fetch each argument passed to the command line.

Syntax

Its syntax is as follows:

```
var system = require('system');
system.args;
```

Example

Take a look at the following example to understand how this property works.

```
var system = require('system');
console.log(system.args.length);
if (system.args.length>0) {
    console.log(JSON.stringify(system.args));
}
phantom.exit();
```

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

Command: phantomsjs args.js <http://localhost/tasks/request.html>

```
2
["args.js","http://localhost/tasks/request.html"]
```

The first value is the name of the file and next value is the URL passed. You can pass more arguments in the command line and fetch the same using **system.args**.

env

The **env** property returns the details of the system.

Syntax

Its syntax is as follows:

```
var system = require('system');
system.env;
```

Example

```
var system = require('system');
console.log(JSON.stringify(system.env));
phantom.exit();
```

The above program generates the following output.

```
{":":":::\\", "=C:":"C:\\phantomjs\\bin", "=ExitCode":"00000000", "ALLUSERSPROFILE":"C:\\ProgramData", "APPDATA":"C:\\Users\\UserName\\AppData\\Roaming", "COMPUTERNAME":"X", "ComSpec":"C:\\Windows\\system32\\cmd.exe", "CommonProgramFiles":"C:\\Program Files (x86)\\Common Files", "CommonProgramFiles(x86)":"C:\\Program Files (x86)\\Common Files", "CommonProgramW6432":"C:\\Program Files\\Common Files", "FP_NO_HOST_CHECK":"NO", "HOMEDRIVE":"C:", "HOMEPATH":"\\Users\\UserName", "LOCALAPPDATA":"C:\\Users\\UserName\\AppData\\Local", "LOGONSERVER":"\\\\MicrosoftAccount", "NUMBER_OF_PROCESSORS":"2", "OS":"Windows_NT", "PATHEXT": ".COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC", "PROCESSOR_ARCHITECTURE":"x86", "PROCESSOR_ARCHITECTUREW6432":"AMD64", "PROCESSOR_IDENTIFIER":"Intel64 Family 6 Model 58Stepping9, GenuineIntel", "PROCESSOR_LEVEL":"6", "PROCESSOR_REVISION":"3a09", "PROMPT":"$P$G", "PSModulePath":"C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\Modules\\", "PUBLIC":"C:\\Users\\Public", "Path":"C:\\Program Files\\Dell\\DW WLAN Card;c:\\Program Files (x86)\\Intel\\iCLS Client\\;c:\\Program Files\\Intel\\iCLSClient\\;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\Program Files\\Intel\\Intel(R) Management Engine Components\\DAL;C:\\Program Files\\Intel\\Intel(R) Management Engine Components\\IPT;C:\\Program Files (x86)\\Intel\\Intel(R) Management Engine Components\\DAL;C:\\Program Files (x86)\\Intel\\Intel(R) Management Engine Components\\IPT;c:\\ProgramFiles\\WIDCOMM\\BluetoothSoftware\\;c:\\ProgramFiles\\WIDCOMM\\BluetoothSoftware\\syswow64;C:\\ProgramFiles(x86)\\WindowsLive\\Shared;C:\\ProgramFiles\\nodejs\\;C:\\ProgramFiles\\Git\\cmd;C:\\ProgramFiles\\OpenVPN\\bin;C:\\ProgramFiles(x86)\\Skype\\Phone\\;C:\\Users\\UserName\\AppData\\Roaming\\npm", "ProgramData":"C:\\ProgramData", "ProgramFiles":"C:\\ProgramFiles(x86)", "ProgramFiles(x86)":"C:\\ProgramFiles(x86)", "ProgramW6432":"C:\\ProgramFiles", "SESSIONNAME":"Console", "SystemDrive":"C:", "SystemRoot":"C:\\Windows", "TEMP":"C:\\Users\\Username~1\\AppData\\Local\\Temp", "TMP":"C:\\Users\\Username~1\\AppData\\Local\\Temp", "USERDOMAIN":"USER", "USERDOMAIN_ROAMINGPROFILE":"USER", "USERNAME":"X Y", "USERPROFILE":"C:\\Users\\X Y", "windir":"C:\\Windows"}
```

OS

It returns the details of the operating system used. It returns an object with architecture, name of the OS, and version.

Syntax

Its syntax is as follows:

```
var system = require('system');  
system.os;
```

Example

```
var system = require('system');  
console.log(JSON.stringify(system.os));  
phantom.exit();
```

The above program generates the following output.

```
{"architecture":"32bit","name":"windows","version":"8.1"}
```

pid

This property returns the process ID.

Syntax

Its syntax is as follows:

```
var system = require('system');  
system.pid;
```

Example

Let us look at an example of the **pid** property.

```
var system = require('system');  
console.log(system.pid);  
phantom.exit();
```

The above program generates the following output.

```
2160
```

platform

This property returns the platform we are working on.

Syntax

Its syntax is as follows:

```
var system = require('system');  
system.platform;
```

Example

```
var system = require('system');  
console.log(system.platform);  
phantom.exit();
```

The above program generates the following output.

```
Phantomjs
```

Web Server Module

12. PhantomJS – Properties

PhantomJS uses an embedded web server called **mongoose**. Right now, PhantomJS cannot connect with any other production webserver. With respect to connection, it can deal with 10 connections at a time and more than 10 requests will be waiting in a queue.

To start a webserver, we need to use the following syntax:

```
var webserver = require ('webserver');
```

Let us understand the **Port** property, which is used to listen to the requests that are sent to the webserver.

port

The Port property for a webserver is used to listen to the request that are sent to it.

Syntax

Its syntax is as follows:

```
var server = require('webserver').create();  
var listening = server.listen(port, function (request, response) {});
```

Example

Let us take an example to understand how the **port** property works.

```
var webserver = require('webserver');  
var server = webserver.create();  
var service = server.listen(8080,function(request,response){  
});  
if(service) console.log("server started - http://localhost:" + server.port);
```

The above program generates the following output.

```
server started - http://localhost:8080
```


13. PhantomJS – Methods

In this chapter, we will discuss regarding the various methods of the Web Server Module of PhantomJS.

close

The **close** method is used to close the webserver.

Syntax

Its syntax is as follows:

```
var server = require('webserver').create();
server.close();
```

Example

The following example shows how you can use the **close** method.

```
var webserver = require('webserver');
var server = webserver.create();
var service = server.listen(8080,function(request,response){
});
if(service) console.log("server started - http://localhost:" + server.port);
console.log(server.port);
server.close();
console.log(server.port);
```

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

```
server started - http://localhost:8080
8080
```

Here, we have console'd **server.port** after closing the server. Hence, it will not respond, as the webserver is already closed.

listen

The **server.listen** method takes the port and callback function with two arguments, which are – **Request Object** and **Response Object**.

The **Request Object** contains the following properties:

- **Method:** This defines the method GET /POST.

- **URL:** This displays the requested URL.
- **httpVersion:** This displays the actual http version.
- **Headers:** This displays all the headers with key value pairs.
- **Post:** Request body applicable only for the post method.
- **postRaw:** If the Content-Type header is set to 'application/x-www-form-urlencoded', the original content of the post will be stored in this extra property (postRaw) and then that post will be automatically updated with a URL-decoded version of the data.

The **Response Object** contains the following properties:

- **Headers:** Has all the HTTP headers as key-value pairs. It should be set before calling write for the first time.
- **SetHeader:** This sets a specific header.
- **Header (name):** It returns the value of the given header.
- **StatusCode:** It sets the returned HTTP status code.
- **SetEncoding (encoding):** This is used to convert data given to write(). By default, data will be converted to UTF-8. Indicate "binary" if data is a binary string. Not required if data is a Buffer (e.g. from page.renderBuffer).
- **Write (data):** It sends data for the response body. Can be called multiple times.
- **WriteHead (statusCode, headers):** It sends a response header to the request. The status code is a 3-digit HTTP status code (e.g. 404). The last arguments and headers are the response headers.
- **Close:** It closes the http connection.
- **CloseGracefully:** It is similar to close(), but it makes sure the response headers have been sent first.

Syntax

Its syntax is as follows:

```
var server = require('http').createServer();
server.listen(8080, function (request, response) {});
```

Example

Let us take an example to understand how the **listen** method works.

```
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));
    // we set the headers here
    response.statusCode = 200;
    response.headers = {"Cache": "no-cache", "Content-Type": "text/html"};
    // the headers above will now be sent implicitly
    // now we write the body
    response.write("<html><head><title>Welcone to Phantomjs</title></head>");
    response.write("<body><p>Hello World</body></html>");
    response.close();
});
if (!listening) {
    console.log("could not create web server listening on port " + port);
    phantom.exit();
}
var url = "http://localhost:" + port + "/foo/response.php";
console.log("sending request to :" +url);
page.open(url, function (status) {
    if (status !== 'success') {
        console.log('page not opening');
    } else {
        console.log("Getting response from the server:");
        console.log(page.content);
    }
    phantom.exit();
});
```

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-IN,*",
    "Connection": "Keep-Alive",
    "Host": "localhost:8080",
    "User-Agent": "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/538.1
(KHTML, like Gecko) PhantomJS/2.1.1 Safari/538.1"
  },
  "httpVersion": "1.1",
  "method": "GET",
  "url": "/foo/response.php"
}
Getting response from the server:
<html><head><title>Welcone to Phantomjs</title></head><body><p>Hello
World</p></
body></html>
```

Miscellaneous

14. PhantomJS – Command Line Interface

PhantomJS can be executed using the keyword "phantomjs" and the name of the file. For example, "phantomjs file.js".

You can also pass arguments into "file.js" as follows:

```
phantomjs [options] file.js arg1 arg2 ...
```

Let us go through some of the options available at the command line.

Option	Description
--help or -h	Will display all the command line options. Halts immediately and will not run the script passed
--version or -v	Prints the version of PhantomJS running. This will halt the execution of script to be run.
--debug[true false]	Prints warnings and debug messages. By default, it is set to false. You can also use [yes no].
--cookies-file = /path/to/cookies.txt	File name to store persistent cookies.
--disk-cache=[true false]	Enables disk cache. It also takes values [yes no]
--disk-cache-path	Path for disk cache.
--ignore-ssl-errors=[true false]	Ignore SSL errors. For example – Expired or self-signed certificate errors. Default is false. It also takes values [yes no].
--load-images=[true false]	Loads all inline images. Default value is true. It also takes [yes no]
--local-storage-path=/some/path	Path to save LocalStorage content and WebSQL content.
--local-storage-quota=number	Maximum size to allow for data.
--local-url-access	Allows use of 'file:/// ' URLs (default is true).

<code>--local-to-remote-url-access=[true false]</code>	Allows local content to access remote URL. Default is false. It also takes values [yes no]
<code>--max-disk-cache-size=size</code>	Limits the size of disk cache (in KB)
<code>--offline-storage-path</code>	Specifies location for offline storage.
<code>--offline-storage-quota</code>	Sets the maximum size of the offline storage in KB.
<code>--output-encoding=encoding</code>	Sets the encoding used for terminal output. Default is utf-8.
<code>--proxy=address:port</code>	Specifies the proxy server to be used (For example – proxy=192.168.1.1:8080)
<code>--proxy-type=[http socks5 none]</code>	Specifies the type of proxy server (default is http)
<code>--proxy-auth</code>	Authentication information for the proxy. For example – --proxy-auth=username:password
<code>--script-encoding=encoding</code>	Sets the encoding used for the script. Default is utf8.
<code>--script-language</code>	Sets the script language.

15. PhantomJS – Screen Capture

PhantomJS is very helpful in taking a screenshot of a webpage and converting a webpage into a PDF. We have given here a simple example to demonstrate how it works.

Example

```
var page = require('webpage').create();
page.open('http://phantom.org/',function(status){
    page.render('phantom.png');
    phantom.exit();
});
```

Execute the above program and the output will be saved as **phantom.png**.

The screenshot shows the PhantomJS website homepage. At the top, there's a navigation bar with links for SOURCE CODE, DOCUMENTATION, API, EXAMPLES, and FAQ. The main heading reads "Full web stack No browser required". Below this, a description states: "PhantomJS is a headless WebKit scriptable with a JavaScript API. It has fast and native support for various web standards: DOM handling, CSS selector, JSON, Canvas, and SVG." There are two buttons: "Download v2.1" and "Get started". A code block on the right shows a "Simple Javascript example" for opening a page and rendering it. A "Community" section includes links to "Read the release notes", "Join the mailing list", and "Report bugs". The main content area features four columns: "HEADLESS WEBSITE TESTING", "SCREEN CAPTURE", "PAGE AUTOMATION", and "NETWORK MONITORING", each with a brief description and a "Learn more" link. At the bottom, it mentions that PhantomJS is used in various open-source projects like Bootstrap, CodeMirror, Ember.js, etc. The footer contains copyright information from 2010-2017.

PhantomJS

SOURCE CODE DOCUMENTATION API EXAMPLES FAQ

Full web stack
No browser required

PhantomJS is a headless WebKit scriptable with a JavaScript API. It has fast and native support for various web standards: DOM handling, CSS selector, JSON, Canvas, and SVG.

Download v2.1 Get started

Community: Read the release notes Join the mailing list Report bugs

PhantomJS is an optimal solution for

HEADLESS WEBSITE TESTING
Run functional tests with frameworks such as Jasmine, QUnit, Mocha, Capybara, WebDriver, and many others. [Learn more](#)

SCREEN CAPTURE
Programmatically capture web contents, including SVG and Canvas. Create web site screenshots with thumbnail preview. [Learn more](#)

PAGE AUTOMATION
Access and manipulate webpages with the standard DOM API, or with usual libraries like jQuery. [Learn more](#)

NETWORK MONITORING
Monitor page loading and export as standard HAR files. Automate performance analysis using YSlow and Jenkins. [Learn more](#)

PhantomJS is used in the test workflow of various open-source projects: [Bootstrap](#), [CodeMirror](#), [Ember.js](#), [jQuery Mobile](#), [Less.js](#), [Modernizr](#), [YUI3](#), and [many more](#).

© Copyright 2010-2017 Artya Hidayat — Homepage design by Maurice Syay — Documentation design by Jarve Mason

Convert Webpages into PDFs

PhantomJS also helps to convert webpages into PDFs with header and footer added to it. Take a look at the following example to understand how it works.

```
var wpage = require('webpage').create();
var url = "https://en.wikipedia.org/wiki/Main_Page";
var output = "test.pdf";
wpage.paperSize = {
  width: screen.width+'px',
  height: '1500px',
  margin: {'top':'50px',
    'left':'50px',
    'right':'50px'
  },
  orientation:'portrait',
  header: {
    height: "1cm",
    contents: phantom.callback(function(pageNumber, nPages) {
      return "<h5>Header <b>" + pageNumber + " / " + nPages + "</b></h5>";
    })
  },
  footer: {
    height: "1cm",
    contents: phantom.callback(function(pageNumber, nPages) {
      return "<h5>Footer <b>" + pageNumber + " / " + nPages + "</b></h5>";
    })
  }
}
wpage.open(url, function (status) {
  if (status !== 'success') {
    console.log('Page is not opening');
    phantom.exit();
  } else {
    wpage.render(output);
    phantom.exit();
  }
});
```

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

The above will convert the page into pdf and will be saved in test.pdf

Convert a Canvas to an Image

Phantomjs can easily convert a Canvas to an Image. Take a look at the following example to understand how it works.

```
var page = require('webpage').create();
page.content = '<html><body><canvas id="surface" width="400"
height="400"></canvas></body></html>';
page.evaluate(function() {
    var context,e1;
    e1 = document.getElementById('surface');
    context = e1.getContext('2d');
    context.font="30px Comic Sans MS";
    context.fillStyle = "red";
    context.textAlign = "center";
        context.fillText("Welcome to PhantomJS ", 200, 200);
    document.body.style.backgroundColor = 'white';
        document.body.style.margin = '0px';
});
page.render('canvas.png');
phantom.exit();
```

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

Welcome to PhantomJS

16. PhantomJS – Page Automation

PhantomJS, with the help of its webpage module APIs, can manipulate webpages and perform operations such as DOM manipulation, clicking on buttons, etc.

Fetch Images from a Page

The following program shows how you can use PhantomJS to fetch images from a page.

```
var wpage = require('webpage').create();
wpage.onConsoleMessage = function(str) {
    console.log(str.length);
}
wpage.open("http://phantomjs.org", function(status) {
    console.log(status);
    var element = wpage.evaluate(function() {
        var imgdata = document.querySelectorAll('img');
        var imgsrc = [];
        if (imgdata) {
            for (var i in imgdata) {
                imgsrc.push(imgdata[i].src);
            }
        }
        return imgsrc;
    });
    console.log(JSON.stringify(element));
});
```

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

```
Success
["http://phantomjs.org/img/phantomjs-
logo.png","http://phantomjs.org/img/phantom
js-logo.png","http://phantomjs.org/img/phantomjs-
logo.png","http://phantomjs.org
/img/phantomjs-logo.png"]
```

It is possible to include external JavaScript in a page using the **injectJS webpage method**. There are many properties and methods, which can help page automation and do many other things. You can refer the webpage module where the properties and methods are explained in detail.

17. PhantomJS – Network Monitoring

With help of PhantomJS, we can monitor the network and validate the behavior and performance of a particular webpage. There are callbacks in PhantomJS, i.e., **onResourceRequested** and **onResourceReceived**, which help in monitoring the traffic for a given page.

Example

The following example shows how you can use PhantomJS to monitor traffic for a given page.

```
function createHAR(address, title, startTime, resources) {
// this function formats the data which is coming from onresourcerequest and
onresourcereceived
    var entries = [];
    resources.forEach(function (resource) {
        var request = resource.request,
            startReply = resource.startReply,
            endReply = resource.endReply;

        if (!request || !startReply || !endReply) {
            return;
        }

        // Exclude Data URI from HAR file because
        // they aren't included in specification
        if (request.url.match(/^data:image\/.*$/i)) {
            return;
        }

        entries.push({
            startedDateTime: request.time.toISOString(),
            time: endReply.time - request.time,
            request: {
                method: request.method,
                url: request.url,
                httpVersion: "HTTP/1.1",
                cookies: [],
```

```

        headers: request.headers,
        queryString: [],
        headersSize: -1,
        bodySize: -1
    },
    response: {
        status: endReply.status,
        statusText: endReply.statusText,
        httpVersion: "HTTP/1.1",
        cookies: [],
        headers: endReply.headers,
        redirectURL: "",
        headersSize: -1,
        bodySize: startReply.bodySize,
        content: {
            size: startReply.bodySize,
            mimeType: endReply.contentType
        }
    },
    cache: {},
    timings: {
        blocked: 0,
        dns: -1,
        connect: -1,
        send: 0,
        wait: startReply.time - request.time,
        receive: endReply.time - startReply.time,
        ssl: -1
    },
    pageref: address
    });

    });

    return {
        log: {
            version: '1.2',

```

```

        creator: {
            name: "PhantomJS",
            version: phantom.version.major + '.' + phantom.version.minor +
                '.' + phantom.version.patch
        },
        pages: [{
            startedDateTime: startTime.toISOString(),
            id: address,
            title: title,
            pageTimings: {
                onLoad: page.endTime - page.startTime
            }
        }],
        entries: entries
    }
};
}

var page = require('webpage').create(),
    system = require('system');
var fs = require('fs');

if (system.args.length === 1) {
    console.log('Usage: netsniff.js <some URL>');
    phantom.exit(1);
} else {
    page.address = system.args[1];
    page.resources = [];
    page.onLoadStarted = function () {          // called when page is loaded
        page.startTime = new Date();
    };

    page.onResourceRequested = function (req) {
        // called when any files are requested from given page url
        page.resources[req.id] = {
            request: req,

```

```

        startReply: null,
        endReply: null
    };
};

page.onResourceReceived = function (res) {
//called when any files are received.
    if (res.stage === 'start') {
        page.resources[res.id].startReply = res;
    }
    if (res.stage === 'end') {
        page.resources[res.id].endReply = res;
    }
};

page.open(page.address, function (status) {    // open given page url
    var har;
    if (status !== 'success') {
        console.log('FAIL to load the address');
        phantom.exit(1);
    } else {
        page.endTime = new Date();
        page.title = page.evaluate(function () { // gets the page title
            return document.title;
        });
        har = createHAR(page.address, page.title, page.startTime, page.resources);
        // calls the function createHAR with page url, starttime, and page resources.
        // console.log(JSON.stringify(har, undefined, 4));
        fs.write('log.txt', JSON.stringify(har, undefined, 4), 'w');
        // logs are collected in log.txt file.
        phantom.exit();
    }
});
}

```

Example of log.txt given to HAR preview


```

{
  "log": {
    "version": "1.2",
    "creator": {
      "name": "PhantomJS",
      "version": "2.1.1"
    },
    "pages": [
      {
        "startedDateTime": "2017-05-21T13:41:21.824Z",
        "id": "http://www.sample.com",
        "title": "Free Sample Products - Sample.com » Free Samples,
Free Product Samples, Product Test Marketing",
        "pageTimings": {
          "onLoad": 11081
        }
      }
    ],
    "entries": [
      {
        "startedDateTime": "2017-05-21T13:41:21.815Z",
        "time": 1999,
        "request": {
          "method": "GET",
          "url": "http://www.sample.com/",
          "httpVersion": "HTTP/1.1",
          "cookies": [],
          "headers": [
            {
              "name": "Accept",
              "value":
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
            },
            {
              "name": "User-Agent",
              "value": "Mozilla/5.0 (Windows NT 6.2; WOW64)
AppleWebKit/538.1 (KHTML, like Gecko) PhantomJS/2.1.1 Safari/538.1"
            }
          ]
        }
      }
    ]
  }
}

```

```
    }  
  ],  
  "queryString": [],  
  "headersSize": -1,  
  "bodySize": -1  
},  
"response": {  
  "status": 301,  
  "statusText": "Moved Permanently",  
  "httpVersion": "HTTP/1.1",  
  "cookies": [],  
  "headers": [  
    {  
      "name": "Date",  
      "value": "Sun, 21 May 2017 13:41:25 GMT"  
    },  
    {  
      "name": "Server",  
      "value": "Apache/2.2.14 (Ubuntu)"  
    },  
    {  
      "name": "Location",  
      "value": "http://sample.com/"  
    },  
    {  
      "name": "Vary",  
      "value": "Accept-Encoding"  
    },  
    {  
      "name": "Content-Encoding",  
      "value": "gzip"  
    },  
    {  
      "name": "Keep-Alive",  
      "value": "timeout=15, max=100"  
    },  
  ],  
}
```

```

        {
            "name": "Connection",
            "value": "Keep-Alive"
        },
        {
            "name": "Content-Type",
            "value": "text/html; charset=iso-8859-1"
        }
    ],

    "redirectURL": "",
    "headersSize": -1,
    "bodySize": 307,
    "content": {
        "size": 307,
        "mimeType": "text/html; charset=iso-8859-1"
    }
},
"cache": {},
"timings": {
    "blocked": 0,
    "dns": -1,
    "connect": -1,
    "send": 0,
    "wait": 1999,
    "receive": 0,
    "ssl": -1
},
"pageref": "http://www.sample.com"
},
{
    "startedDateTime": "2017-05-21T13:41:24.898Z",
    "time": 885,
    "request": {
        "method": "GET",
        "url": "http://sample.com/",

```

```

        "httpVersion": "HTTP/1.1",
        "cookies": [],
        "headers": [
            {
                "name": "Accept",
                "value":
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
            },
            {
                "name": "User-Agent",
                "value": "Mozilla/5.0 (Windows NT 6.2; WOW64)
AppleWebKit/538.1 (KHTML, like Gecko) PhantomJS/2.1.1 Safari/538.1"
            }
        ],
        "queryString": [],
        "headersSize": -1,
        "bodySize": -1
    },
    "response": {
        "status": 200,
        "statusText": "OK",
        "httpVersion": "HTTP/1.1",
        "cookies": [],
        "headers": [
            {
                "name": "Date",
                "value": "Sun, 21 May 2017 13:41:27 GMT"
            },
            {
                "name": "Server",
                "value": "Apache/2.2.14 (Ubuntu)"
            },
            {
                "name": "X-Powered-By",
                "value": "PHP/5.3.2-1ubuntu4.29"
            },
            {

```

```

        "name": "X-Pingback",
        "value": "http://sample.com/xmlrpc.php"
    },
    {
        "name": "Link",
        "value": "<http://sample.com/wp-json/>;
rel=\"https://api.w.org/\", <http://wp.me/P6Jj5H-4>; rel=shortlink"
    },
    {
        "name": "Vary",
        "value": "Accept-Encoding"
    },
    {
        "name": "Content-Encoding",
        "value": "gzip"
    },
    {
        "name": "Keep-Alive",
        "value": "timeout=15, max=99"
    },
    {
        "name": "Connection",
        "value": "Keep-Alive"
    },
    {
        "name": "Content-Type",
        "value": "text/html; charset=UTF-8"
    }
],
"redirectURL": "",
"headersSize": -1,
"bodySize": 1969,
"content": {
    "size": 1969,
    "mimeType": "text/html; charset=UTF-8"
}

```

```

    }
  },
  "cache": {},
  "timings": {
    "blocked": 0,
    "dns": -1,
    "connect": -1,
    "send": 0,
    "wait": 869,
    "receive": 16,

    "ssl": -1
  },
  "pageref": http://www.sample.com
},
{
  "startedDateTime": "2017-05-21T13:41:25.767Z",
  "time": 388,
  "request": {
    "method": "GET",
    "url": "http://sample.com/wp-content/themes/samplecom/style.css",
    "httpVersion": "HTTP/1.1",
    "cookies": [],
    "headers": [
      {
        "name": "Accept",
        "value": "text/css,*/*;q=0.1"
      },
      {
        "name": "Referer",
        "value": "http://sample.com/"
      },
      {
        "name": "User-Agent",
        "value": "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/538.1 (KHTML, like Gecko) PhantomJS/2.1.1 Safari/538.1"
      }
    ]
  }
}

```

```

    }
  ],
  "queryString": [],
  "headersSize": -1,
  "bodySize": -1
},
"response": {
  "status": 200,
  "statusText": "OK",
  "httpVersion": "HTTP/1.1",
  "cookies": [],
  "headers": [
    {
      "name": "Date",
      "value": "Sun, 21 May 2017 13:41:27 GMT"
    },
    {
      "name": "Server",
      "value": "Apache/2.2.14 (Ubuntu)"
    },
    {
      "name": "Last-Modified",
      "value": "Fri, 22 Apr 2011 00:32:22 GMT"
    },
    {
      "name": "ETag",
      "value": "\"e1d7-1836-4a176fdbbd180\""
    },
    {
      "name": "Accept-Ranges",
      "value": "bytes"
    },
    {
      "name": "Vary",
      "value": "Accept-Encoding"
    }
  ]
}

```

```
    },
    {
      "name": "Content-Encoding",
      "value": "gzip"
    },
    {
      "name": "Keep-Alive",
      "value": "timeout=15, max=98"
    },
    {
      "name": "Connection",
      "value": "Keep-Alive"
    },
    {
      "name": "Content-Type",
      "value": "text/css"
    }
  ],

  "redirectURL": "",
  "headersSize": -1,
  "bodySize": 3174,
  "content": {
    "size": 3174,
    "mimeType": "text/css"
  }
},
"cache": {},
"timings": {
  "blocked": 0,
  "dns": -1,
  "connect": -1,
  "send": 0,
  "wait": 388,
  "receive": 0,
  "ssl": -1
```



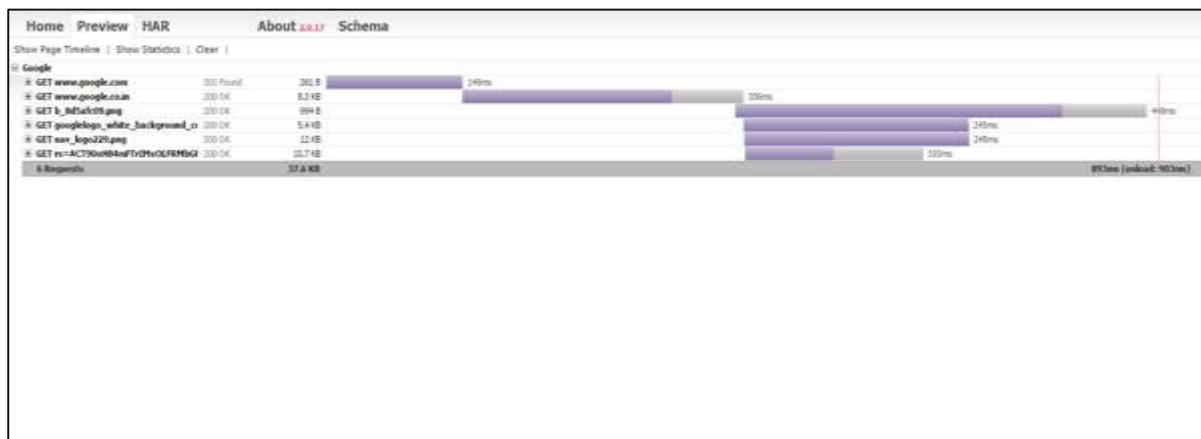
```

    },
    "pageref": "http://www.sample.com"
  }
}
}

```

Output

The above program generates the following output.



All the logs are collected in a **.txt file** and later, the same is converted as above using the HAR preview, for example, (<http://www.softwareishard.com/har/viewer/>)

18. PhantomJS – Testing

PhantomJS has a lot of API for a webpage, which gives all the details. PhantomJS can be used to for testing like getting the content of a page, taking screen share, converting page into a pdf, etc. There are many popular testing libraries in the market, which can be used along with PhantomJS and can be used for testing.

Some of the popular frameworks that can work with PhantomJS are as follows:

- Mocha
- Jasmine
- Qunit
- Hiro
- Laika
- Buster.JS
- WebDriver

Example – PhantomJS with Qunit

```
(function () {  
    var url, timeout, args = require('system').args, page = require('webpage').create();  
    url = args[1];  
    timeout = parseInt(10, 10);  
    page.onConsoleMessage = function (msg) {  
        //prints all the console messages  
        console.log(msg);  
    };  
    page.onInitialized = function () { // called when page is initialized  
        page.evaluate(callqunit);  
    };  
    page.onCallback = function (message) { // called from  
        var result, failed;  
        if (message) {  
            if (message.name === 'QUnit.done') {  
                result = message.data;  
                failed = !result || !result.total || result.failed;  
  
                if (!result.total) {
```

```

        console.error('No tests were executed');
    }
    pageexit(failed ? 1 : 0);
}
};

page.open(url, function (status) { // opening page
    if (status !== 'success') {
        console.error('Unable to access network: ' + status);
        pageexit(1);
    } else {
        var checkqunit = page.evaluate(function () {
            //evaluating page and checking if qunit object is present on the
given page url
            return (typeof QUnit === 'undefined' || !QUnit);
        });
        if (checkqunit) {
            console.error('Qunit scripts are not present on the page');
            pageexit(1);
        }

        //timeout of 10seconds is used otherwise message from console will
get printed.
        setTimeout(function () {
            console.error('The specified timeout of ' + timeout + ' seconds
has expired. Aborting...');
            pageexit(1);
        }, timeout * 1000);
    }
});

function callqunit() {

```

qunit.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="https://code.jquery.com/qunit/qunit-2.3.2.css">
</head>
<body>
  <div id="qunit"></div>
  <div id="qunit-fixture"></div>
  <script src="https://code.jquery.com/qunit/qunit-2.3.2.js"></script>
</body>
</html>
```

Output

Command: phantomjs qunit.js http://localhost/tasks/qunit.html

The above program generates the following output.

```
{"passed":3,"failed":2,"total":5,"runtime":23}
Time taken is 23ms to run 5 tests.
3 passed, 2 failed.
```

19. PhantomJS – REPL

REPL stands for **Read Eval Print Loop**. In PhantomJS, REPL is an interactive mode to test the JavaScript code. You can do the same thing, which is done in Google Chrome Inspector or Firebug to execute some piece of code directly on the console. REPL returns you the same platform to execute the scripts.

The typed command is sent to the interpreter for immediate interpretation (EVAL) and to provide feedback (PRINT). Enter **PhantomJS** in the command line and it will take you to the interactive mode, where you can execute your code directly.

Syntax

Its syntax is as follows:

```
Phantomjs
```

Example

The following example demonstrates how REPL works in PhantomJS.

```
phantomjs> console.log("Welcome to phantomjs");
Welcome to phantomjs
Undefined

phantomjs> window.navigator
{
  "appCodeName": "Mozilla",
  "appName": "Netscape",
  "appVersion": "5.0 (Windows NT 6.2; WOW64) AppleWebKit/538.1 (KHTML, like Gecko)
PhantomJS/2.1.1 Safari/538.1",
  "cookieEnabled": true,
  "language": "en-IN",
  "mimeTypes": {
    "length": 0
  },
  "onLine": false,
  "platform": "Win32",
  "plugins": {
    "length": 0
  },
  "product": "Gecko",
```

```
"productSub": "20030107",
  "userAgent": "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/538.1 (KHTML,
like Gecko) PhantomJS/2.1.1 Safari/538.1",
  "vendor": "Apple Computer, Inc.",
  "vendorSub": ""
}
phantomjs>
```

To find the version

```
phantomjs> phantom.version
{
  "major": 2,
  "minor": 1,
  "patch": 1
}
phantomjs>
```

Each command is executed and the result is displayed. Use CTRL+C, CTRL+D or **phantom.exit()** to come out of the interactive mode.

Use the **up/down** arrow keys to listen to the previously typed commands.

There is another feature called **autocompletion**, which helps to remember the command. Just type "phantom" and hit the "Tab" button to get a list of available commands you can execute.

Output

The above program generates the following output.

```
phantomjs> phantom.→|
phantomjs> phantom.cookies→|
phantomjs> phantom.exit→|
phantomjs> phantom.version→|
```

20. PhantomJS – Examples

In this chapter, we are providing a few more practical examples to understand some important features of PhantomJS.

Example 1 – Find the Page Speed

In this example, we will use PhantomJS to find the **page speed** for any given page URL.

```
var page = require('webpage').create(),
    system = require('system'),
    t, address;

if (system.args.length === 1) {
    console.log('Usage: loadspeed.js <some URL>');
    phantom.exit(1);
} else {
    t = Date.now();
    address = system.args[1];
    page.open(address, function (status) {
        if (status !== 'success') {
            console.log('FAIL to load the address');
        } else {
            t = Date.now() - t;
            console.log('Page title is ' + page.evaluate(function () {
                return document.title;
            }));
            console.log('Loading time ' + t + ' msec');
        }
        phantom.exit();
    });
}
```

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

Command: phantomjs pagespeed.js <http://www.google.com>

```
Page title is Google
Loading time 1396 msec
```

Example 2 — Send a Click Event to a Page

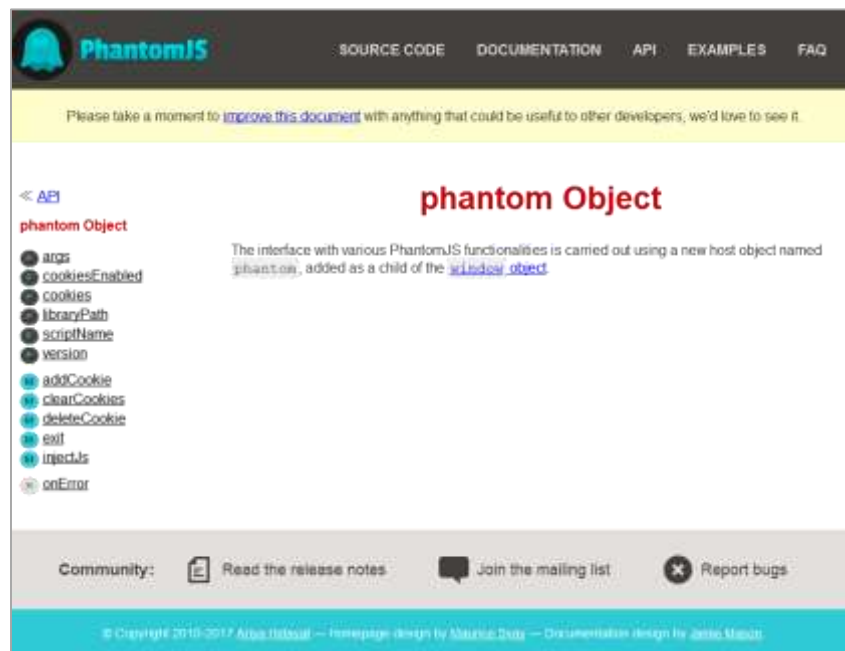
In the following example, we will use PhantomJS to send a **click event** to a page.

```
var page = require('webpage').create();
page.onConsoleMessage = function(str) {
    console.log(str);
}
page.open('http://phantomjs.org/api/phantom/', function(status) {
page.render('beforeclick.png');
console.log(page.url);
var element = page.evaluate(function() {
    return document.querySelector('img[src="http://phantomjs.org/img/phantomjs-
logo.png"]');
});
page.sendEvent('click', element.offsetLeft, element.offsetTop, 'left');
window.setTimeout(function () {
    console.log(page.url);
    page.render('afterclick.png');
    phantom.exit();
}, 5000);
console.log('element is ' + element);
});
```

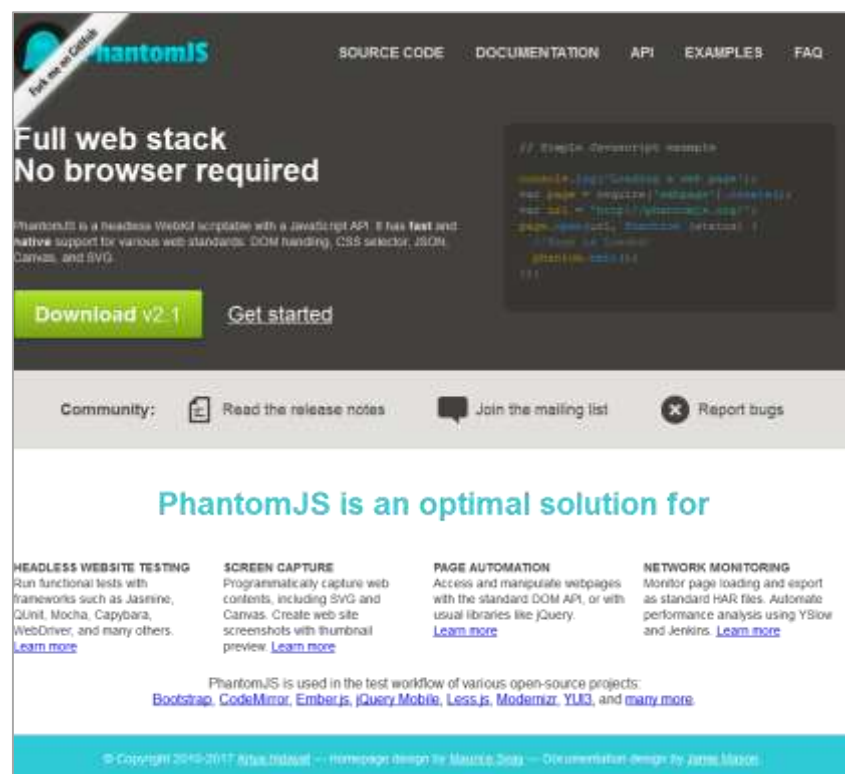
The above program generates the following **output**.

```
http://phantomjs.org/api/phantom/
element is [object Object]
http://phantomjs.org/
```


Our program will create the following two **png** images in the **bin** folder. These two images show the difference before and after the execution of the above program.



beforeclick.png



afterclick.png

Example 3 — Submit a Form

The following example shows how to submit a form using PhantomJS.

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

wpage.open("http://localhost/tasks/submitform.html", function(status) {
    console.log(status);
    wpage.uploadFile('input[name=fileToUpload]', 'output.png');
    wpage.render("sform.png");
    var element = wpage.evaluate(function() {
        return document.querySelector('input[type="submit"]');
        // getting details of submit button using queryselector.
    });
    wpage.sendEvent('click', element.offsetLeft, element.offsetTop, 'left');
    // sendevent is used to send click event and also giving the left and top
    position of the submit button.
    window.setTimeout(function () {
        console.log(wpage.url);
        wpage.render("submit.png"); // screenshot is saved in submit.png
        phantom.exit();
    }, 5000);
    console.log('element is ' + element);
});
```

submitform.html

The following code shows how to use the **submitform.html** file.

```
<html>
<head><title>Window 2</title></head>
<body>
<form action="submitform.php" method="post" enctype="multipart/form-data" id="form1">
    <input type="file" name="fileToUpload" id="fileToUpload">
    <input type="submit" value="Upload Image" name="submit">
</form>
</body>
</html>
```

Once the form is submitted, it goes to **submitform.php**.

submitform.php

submitform.php is just printing the details of the files.

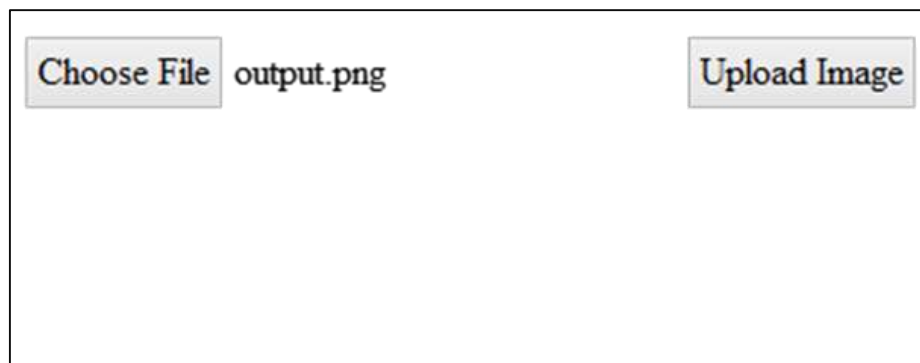
```
<?php
    print_r($_FILES);
?>
```

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

```
Success
element is [object Object]
http://localhost/tasks/submitform.php
```

Images

Following are the images for **file upload** and **form submit**.



File Upload

```
Array ( [fileToUpload] => Array ( [name] => output.png [type]
=> image/png [tmp_name] => C:\xampp\tmp\php65E8.tmp
[error] => 0 [size] => 8286 ) )
```

Form Submit