

Headless Chrome Automation with

THE CRRRI PACKAGE

Romain Lesur



Deputy Head of the
Statistical Service



Retrouvez-nous sur
justice.gouv.fr



MINISTÈRE
DE LA JUSTICE

Web browser

A web browser is like a shadow puppet theater



Suyash Dwivedi
[CC BY-SA 4.0](#)
[via Wikimedia Commons](#)

Behind the scenes

The puppet masters



Mr.Niwat Tantayanusorn,
Ph.D.
[CC BY-SA 4.0](#)
[via Wikimedia Commons](#)

What is a headless browser?

- Turn off the light: no visual interface
- Be the stage director... in the dark! 🤭



Kent Wang from London, United Kingdom
[CC BY-SA 2.0](#)
[via Wikimedia Commons](#)

Some use cases

- Responsible web scraping
(with JavaScript generated content)
- Webpages screenshots
- PDF generation
- Testing websites (or Shiny apps)

Related packages

Headless browser is an old topic

- {RSelenium} client for Selenium WebDriver, requires a Selenium server (Java).
- {webshot}, {webdriver} relies on the abandoned PhantomJS library.
- {hrbrmstr/htmlunit} uses the `HtmlUnit` Java library.
- {hrbrmstr/splashr} uses the `splash` python library.
- {hrbrmstr/decapitated} uses headless Chrome command-line instructions or the Node.js `gepetto` module (built-on top of the `puppeteer` Node.js module)

Headless Chrome

Since Chrome 59

The crrri package
developed with
Christophe Dervieux
WIP

github.com/RLesur/crrri

- Basic tasks can be executed using command-line instructions
- Offers the possibility to have the full control of Chrome using Node.js modules `puppeteer`, `chrome-remote-interface`...
- Have the full control of  from  **without Java, Node or any server**
- Low-level API inspired by the [chrome-remote-interface](#) JS module **give access to 500+ functions to control Chrome**
- Dedicated to advanced uses / R packages developers
- Compatible with Opera, EdgeHTML and Safari

Technical explanations

Steps to interact with headless Chrome

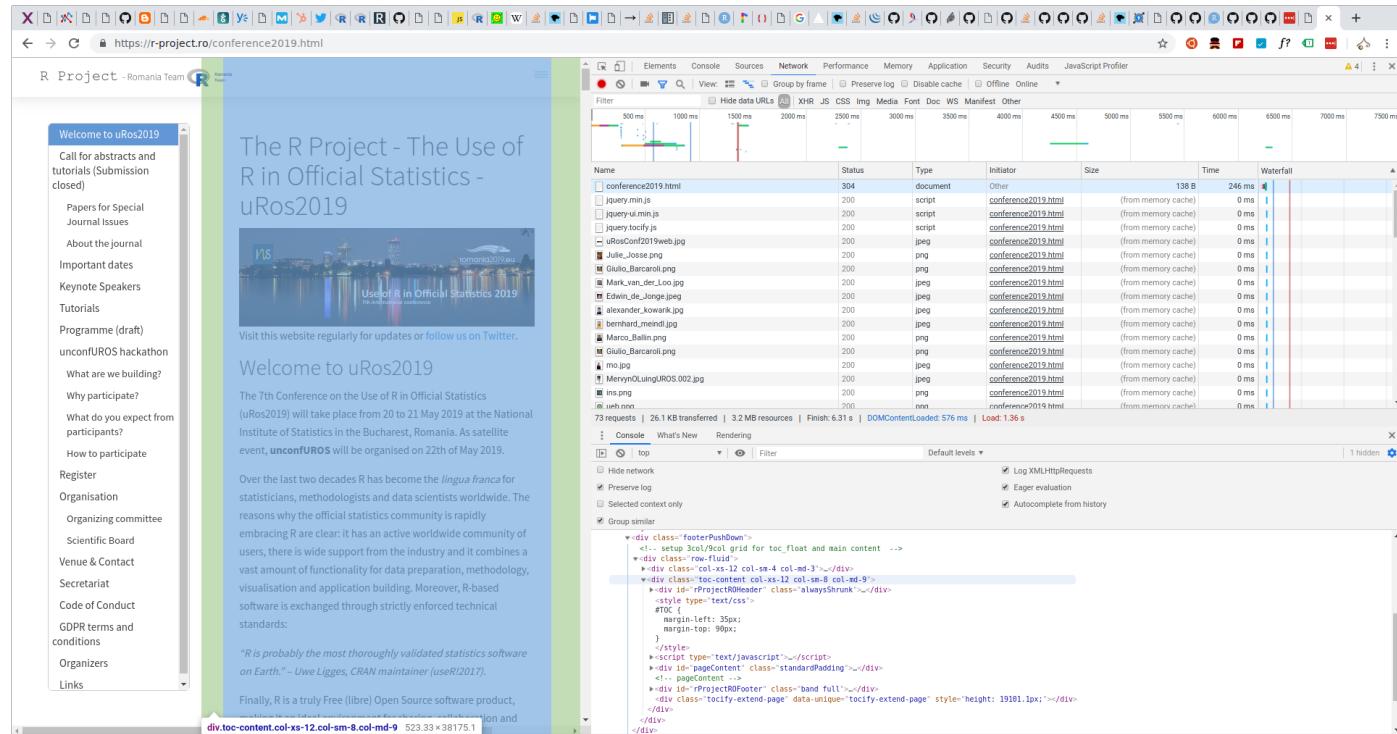
Headless Chrome can be controlled using the **Chrome DevTools Protocol (CDP)**

1. Launch Chrome in headless mode
2. Connect R to Chrome through websockets
3. Build an asynchronous function that
 - sends CDP commands to Chrome
 - listen to CDP events from Chrome
4. Execute this async flow with R

The goal of {crrri} is to **ease these steps**.

Chrome DevTools Protocol

Program actions
usually done with
Chrome DevTools

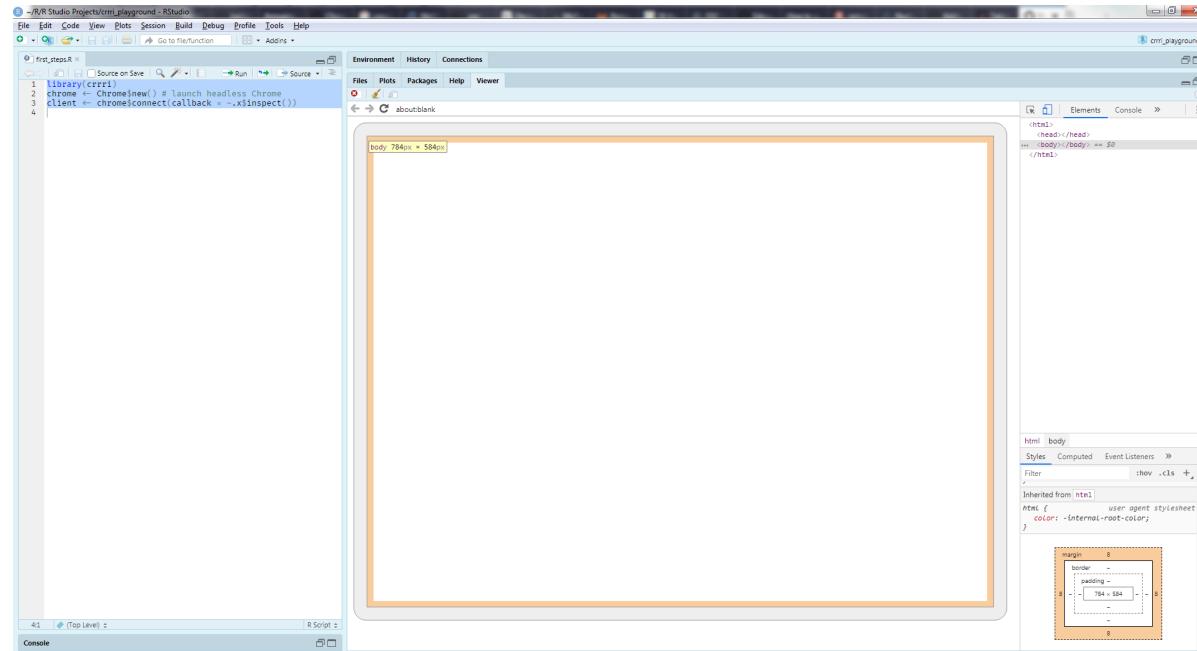


Playing with headless Chrome in RStudio 1.2

client is a connection object

Inspect headless Chrome in RStudio

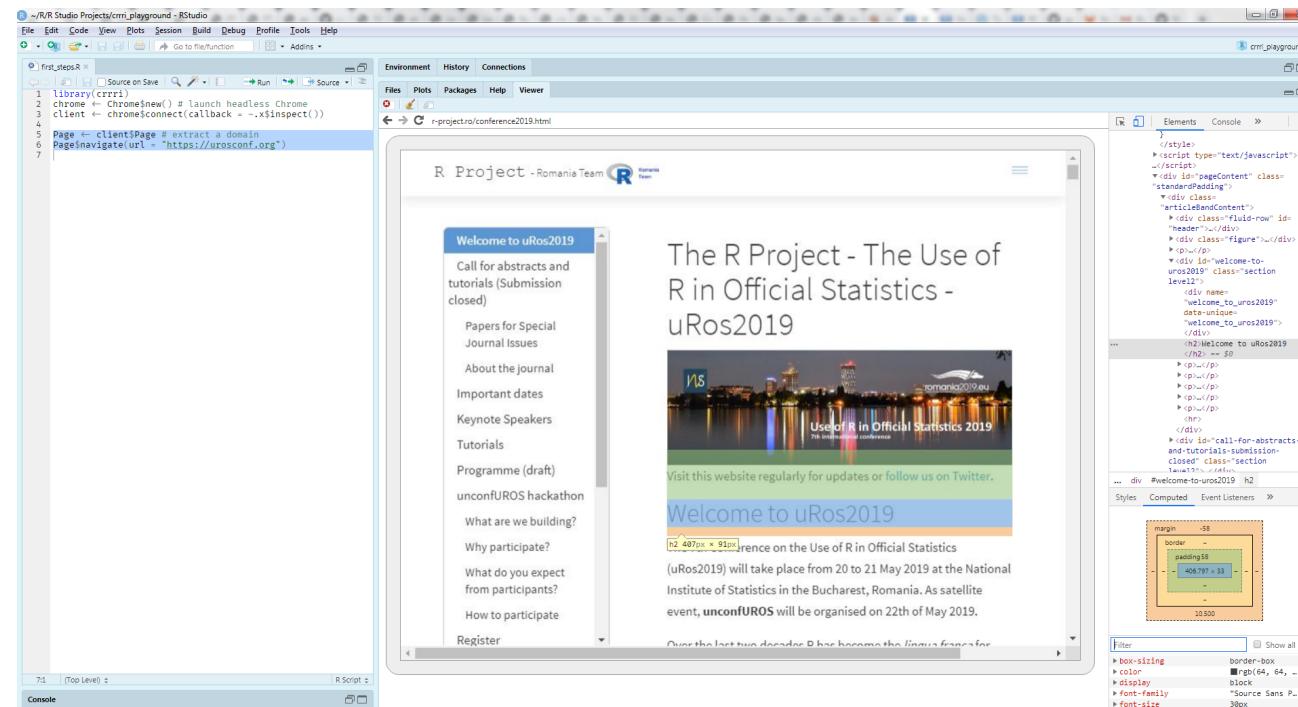
```
remotes::install_github("rlesur/crrri")
library(crrri)
chrome <- Chrome$new() # launch headless Chrome
client <- chrome$connect(callback = ~.x$inspect())
```



Chrome DevTools Protocol commands: an example

A domain is a set of commands and events listeners

```
Page <- client$Page # extract a domain  
Page$navigate(url = "https://urosconf.org")  
#> <Promise [pending]>
```



An API similar to JavaScript

All the functions are asynchronous

```
Page$.navigate(url = "https://urosconf.org")  
#> <Promise [pending]>
```

An object of class Promise from the {promises} package.

Chain with the appropriate pipe!

```
Page$.navigate(url = "https://urosconf.org") %...>%  
  print()  
#> $frameId  
#> [1] "D1660E2ECC76A8356F78820F410BAA8C"  
  
#> $loaderId  
#> [1] "18180FE5BE9D9A60CC37F01610227729"
```

Chaining commands and events listeners

To receive events from Chrome most of domains need to be enabled

```
# ask Chrome to send Page domain events
Page$enable() %...>% {
  # send the 'Page.navigate' command
  Page$navigate(url = "https://urosconf.org")
} %...>% {
  cat('Navigation starts in frame', .frameId, '\n')
  # wait the event 'Page.frameStoppedLoading'
  # fires for the main frame
  Page$frameStoppedLoading(frameId = .frameId)
} %...>% {
  cat('Main frame loaded.\n')
}
```

Chrome DevTools Protocol documentation

chromedevtools.github.io/devtools-protocol

Building higher level functions

Write an asynchronous remote flow

```
print_pdf <- function(client) {  
  Page <- client$Page  
  
  Page$enable() %...>% {  
    Page$navigate(url = "https://r-project.org/")  
    Page$loadEventFired() # await the load event  
  } %...>% {  
    Page$printToPDF()  
  } %...>% # await PDF reception  
  write_base64("r_project.pdf")  
}
```

Modify this script depending on the page content (JS libraries...)

```
perform_with_chrome(print_pdf)
```

Perform this flow synchronously in R

Headless Chrome features

Frequent updates

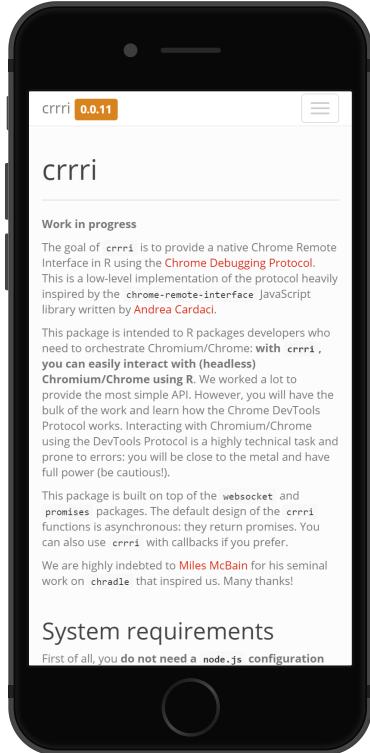
- More than 40 domains, 400+ commands, 100+ events
- Each release of Chrome brings new features.
No need to update {crrri}: commands are fetched from Chrome.

Features

- DOM/CSS manipulation, extraction
- JavaScript Runtime (more than V8)
- Inspect/intercept network traffic
- Emulate devices
- Set JS bindings between Chrome and R
- PDF generation
- Screenshots
- Screencast...

Example: emulate a device

Screenshot your website
with different devices



```
iPhone8 <- function(client) {  
  Emulation <- client$Emulation  
  Page <- client$Page  
  Emulation$setDeviceMetricsOverride(  
    width = 375, height = 667,  
    mobile = TRUE, deviceScaleFactor = 2  
  ) %...>% {  
    Page$enable()  
  } %...>% {  
    Page$navigate("https://rlesur.github.io/crrri")  
  } %...>% {  
    Page$loadEventFired()  
  } %>%  
  wait(3) %...>% {  
    Page$captureScreenshot()  
  } %...>%  
  write_base64("iphone8.png")  
}  
  
perform_with_chrome(iPhone8)
```

Example: screencast

Navigate in RStudio 1.2
and record screencast

[see on Youtube](#)

The screenshot shows the RStudio interface with the following details:

- Code Editor:** A script named `first_steps.R` is open, containing R code for recording a screencast. The code uses the `crrri` package to interact with a browser page (`https://urosconf.org/`) and record its content.
- Browser View:** A browser window displays the "Keynote Speakers" section of a conference website for "R Project - Romania Team". It shows a profile picture of Julie Josse and her bio.
- Environment Tab:** The "Keynote Speakers" tab is selected in the dropdown menu.
- Elements Tab:** The browser's developer tools are open, showing the HTML structure of the "Keynote Speakers" page.
- Console Tab:** The R console is visible at the bottom of the RStudio interface.

Example: web scraping

```
dump_DOM <- function(client) {  
  Page <- client$Page  
  Runtime <- client$Runtime  
  Page$enable() %...>% {  
    Page$navigate(url = 'https://github.com')  
  } %...>% {  
    Page$loadEventFired()  
  } %>%  
  wait(3) %...>% {  
    Runtime$evaluate(  
      expression = 'document.documentElement.outerHTML'  
    )  
  } %...>% {  
    writeLines(.$result$value, "gh.html")  
  }  
}  
  
perform_with_chrome(dump_DOM)
```

Conclusion

Pros

`{crrri}` package

- only one dependency: Chrome
Java, Node or a server are not required
- easy to use with Travis or Docker
- integrates well with RStudio 1.2
- just update Chrome to get the latest features
- flexible: define your own flow
- compatible with Shiny (because of the `{promises}` package)
orchestrate headless Chrome in your Shiny app!

Cons

- low-level interface: Chrome DevTools Protocol is highly technical
- mostly dedicated to R developers/hackers

Credits

Thanks to

- Miles McBain for [chraddle](#)
- Bob Rudis for [decapitated](#)
- Andrea Cardaci for [chrome-remote-interface](#)
- Marvelapp for [devices.css](#)

Questions?

This deck was made with [pagedown](#) and is licensed under 

