Getting Data from the Web with R

Part 6: HTTP Basics and the RCurl Package

Gaston **S**anchez

April-May 2014


http://ww

Readme

License:

You are free to:

Share — copy and redistribute the material

Adapt — rebuild and transform the material

Under the following conditions:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made.

NonCommercial — You may not use this work for commercial purposes.

Share Alike — If you remix, transform, or build upon this work, you must distribute your contributions under the same license to this one.

Lectures Menu

Slide Decks

1. Introduction
2. Reading files from the Web
3. Basics of XML and HTML
4. Parsing XML / HTML content
5. Handling JSON data
6. **HTTP basics and the RCurl package**
7. Getting data via Web forms
8. Getting data via Web services

Web Basics

Goal

HTTP and RCurl

The goal of the present slides is to give you a **crash introduction to HTTP and the RCurl package** so you can be prepared to work with Web Forms (next slides)

Synopsis

In a nutshell

We'll cover the following topics:

- HTTP basics
- HTTP requests and responses
- R package `"RCurl"`

Some References

- HTTP Basics tutorial

  http://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html

- The RCurl Package

  http://www.omegahat.org/RCurl/

- R as a Web Client — the RCurl Package

  http://www.omegahat.org/RCurl/RCurlJSS.pdf

- XML and Web Technlogies for Data Sciences with R
  by Deb Nolan and Duncan Temple Lang

- `libcurl`: the C-URL library

  http://curl.haxx.se/libcurl/

Considerations

Keep in mind

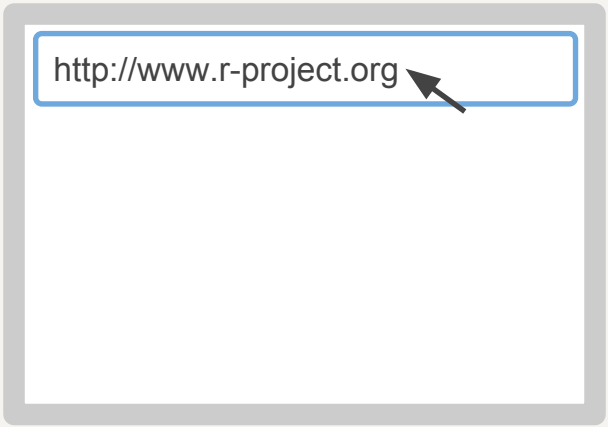
We will discuss preliminary material that will help you understand how to work with data using Web (HTML) Forms:

- Overview of how the Web works
- Basics of **HTTP requests**
- R package `"RCurl"`

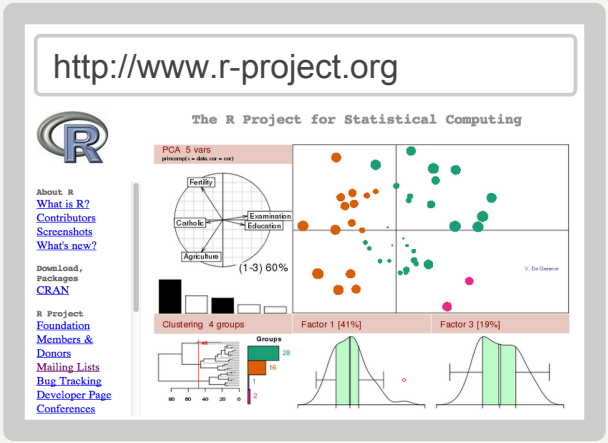
Web Basics

Surfing the Web

1 you open your web browser



2 you type in or click the URL of the website you want to visit



3 the website is displayed in your browser's screen

Surfing the Web

Think about when you surf the web:

- you open a browser
  (eg chrome, mozilla, safari, internet explorer, etc)
- you type in or click the URL of a website you wish to visit
  (eg `http://www.r-project.org`)
- you wait some fractions of a second
- and then the website shows up in your screen

More Formally ...

Surfing the Web

- People access websites using software called a **web browser** (eg Firefox, Chrome, Safari, Internet Explorer)
- The browser in your computer is the **client that requests** a variety of resources (eg pages, images, video, audio, scripts)
- The client's request is sent to **web servers**
- The server **sends responses** back to the client
- The way Clients and Servers dialogue between each other is by following formal **protocols of communication**

The Web


The R Project for Statistical Computing
the
Web
clients with
web browsers
web servers

Basics First

About the Web

- The Web is a massive distributed information system connecting software-and-computers to share information.
- The software-and-computers that form the Web are divided in two types: *clients* and *servers*.
- A **client** is the software that *requests* information (eg html pages, scripts, images, video, audio)
- A **server** is the software-computer in charge of *serving* the resources that the clients request.
- Clients and Servers communicate via the **HyperText Transfer Protocol** (HTTP) and other protocols.

HTTP

HyperText Transfer Protocol

Protocol

*"**Protocol**: a system of rules that explain the correct conduct and procedures to be followed in formal situations"*

http://www.merriam-webster.com/dictionary/protocol

*"A **communications protocol** is a system of digital rules for data exchange within or between computers"*

http://en.wikipedia.org/wiki/Communications_protocol

About HTTP

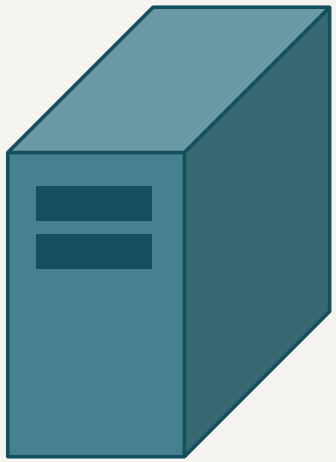
What is HTTP?

- HTTP stands for **HyperText Transfer Protocol**
- HTTP allows clients and servers to communicate (send and receive messages)
- HTTP can be used to transfer documents, images, movies, audio files, data, scripts and all the other web resources that make up websites and applications

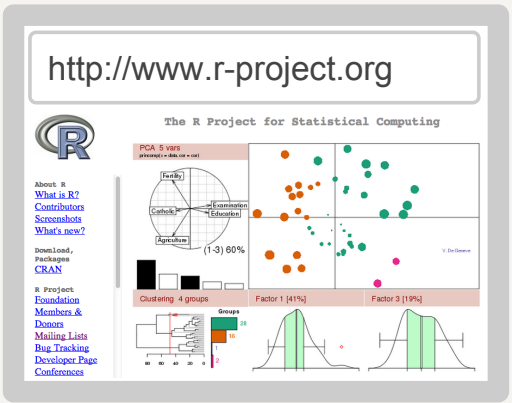
Surfing the Web

1

client's request



web server



web client

server's response

2

Behind the Scenes

Whenever you surf the web:

- your browser sends **HTTP request** messages
  (for HTML pages, images, scripts and styles sheets)
- the HTTP requests are sent to Web servers
- Web servers handle these requests by returning **HTTP response** messages
- the responses contain the requested resource

HTTP Requests and Responses

Suppose we open the browser in order to visit the R project's homepage

HTTP Request and Response

Although we don't see it, there's a client-server dialogue:

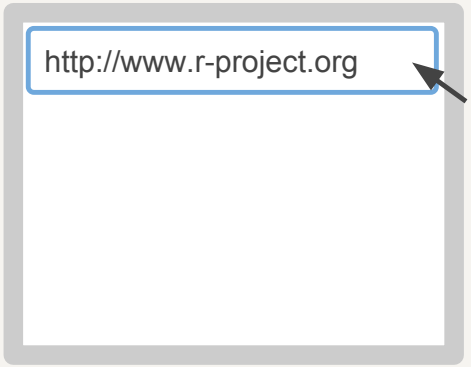
```
GET / HTTP/1.1
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y zlib/1.2.5
Host: r-project.org
Accept: */*

--------------------------------------------------------------------------------

HTTP/1.1 301 Moved Permanently
Date: Thu, 01 May 2014 16:54:43 GMT
Server: Apache/2.2.22 (Debian)
Location: http://www.r-project.org/
Vary: Accept-Encoding
Content-Length: 312
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.r-project.org/">here</a>.</p>
<hr>
<address>Apache/2.2.22 (Debian) Server at r-project.org Port 80</address>
</body></html>
```


Client HTTP Request and Server HTTP Response



1 Client Request → web server

```
GET / HTTP/1.1
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0)
Host: r-project.org
Accept: */*
```

Client ← Server Response 2

```
HTTP/1.1 301 Moved Permanently
Date: Thu, 01 May 2014 16:54:43 GMT
Server: Apache/2.2.22 (Debian)
Location: http://www.r-project.org/
Vary: Accept-Encoding
Content-Length: 312
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
...
</html>
```

HTTP Messages

Anatomy of an HTTP message

HTTP messages consist of 2 parts (separated by a blank line)

1. A message **header**
   - the 1st line in the header is the request/response line
   - the rest of the lines are *headers* formed of `name:value` pairs
2. An optional message **body**

Anatomy of an HTTP Request

The client (your browser) sends a **request** to the server:

```
GET / HTTP/1.1
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y zlib/1.2.5
Host: r-project.org
Accept: */*
```

- The first line is the **request line** which contains:
  `GET / HTTP/1.1`
- The rest of the *headers* are just `name:value` pairs
  eg `Host:  r-project.org`

Anatomy of an HTTP Response

The server sends a **response** to the client:

```
HTTP/1.1 301 Moved Permanently
Date: Thu, 01 May 2014 16:54:43 GMT
Server: Apache/2.2.22 (Debian)
Location: http://www.r-project.org/
Vary: Accept-Encoding
Content-Length: 312
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
...
</html>
```

- The first line is the **status line** which contains:
  `GET / HTTP/1.1`
- The next lines contain **header values**
- The **body** message appears after the blank line
  in this case is the content of the HTML page

HTTP Methods

HTTP Methods

Method	Description
`GET`	retrieves whatever information is identified by the Request-URI
`POST`	request with data enclosed in the request body
`HEAD`	identical to GET except that the server MUST NOT return a message-body in the response
`PUT`	requests that the enclosed entity be stored under the supplied Request-URI
`DELETE`	requests that the origin server delete the resource identified by the Request-URI
`TRACE`	invokes a remote, application-layer loop-back of the request message
`CONNECT`	for use with a proxy that can dynamically switch to being a tunnel

Quick Recap

So far we've seen that:

- The HTTP protocol is a standardized method for transferring data or documents over the Web
- The clients' requests and the servers' responses are handled via the HTTP protocol
- There are 2 types of HTTP messages: **requests** and **responses**
- We don't actually see HTTP messages but they are there *behind the scenes*

R package "RCurl"

Why RCurl?

"The Web is clearly an important source of data for statisticians as is emerging as vital component in distributed computing via Web services. HTTP is the primary mechanism that underlies the Web and data transfer. As such, it is important for programming languages to have tools for HTTP requests and other protocols"

Duncan Temple Lang

http://www.omegahat.org/RCurl/RCurlJSS.pdf

The RCurl Package

RCurl_1.95-4.tar.gz (05 March 2013)

Manual

The RCurl package is an R-interface to the libcurl library that provides HTTP facilities. This allows us to download files from Web servers, post forms, use HTTPS (the secure HTTP), use persistent connections, upload files, use binary content, handle redirects, password authentication, etc.

The primary top-level entry points are

- getURL()
- getURLContent()
- getForm()
- postForm()

However, access to the C-level routines is also available via the R code, and one can specify options to all of the libcurl operations to control how they are performed. Documentation about the options and commands can be found at the libcurl web site

R functions can be specified to collect text from both the response and its headers. This can be used to customize the processing of the requests and feed the results to higher-level processing (e.g. HTML parsing via the htmlTreeParse function in the XML package).

This package will be used to implement the low-level communication in the SSOAP package and other high-level packages that utilize HTTP to exchange requests and data.

Documentation

Package "RCurl"

RCurl

The R package "RCurl" provides the necessary tools for accessing URIs, data and services via HTTP.

Basically, "RCurl" is an R-interface to the C-library libcurl (which provides HTTP facilities)

It is developed by Duncan Temple Lang and the *official* documentation is available at:
http://www.omegahat.org/RCurl

About RCurl

Why RCurl?

R has very basic —*limited*— support for HTTP facilities. But `"RCurl"` provides *steroids* to R for handling HTTP as well as other protocols.

Simply put, `"RCurl"` allows us to **use R as a Web Client**.

About RCurl

RCurl

- provides an R interface to client-side HTTP
- is based on *libcurl*: the cURL C-level library

  http://curl.haxx.se/libcurl
- has support for numerous protocols
  (eg HTTP, HTTPS, FTP, FTPS, TFTP, LDAP)
- provides the infrastructure to build web-based client software

What does RCurl do?

"RCurl" allows us to:

- download URLs
- submit forms in different ways
- supports HTTPS (the secure HTTP)
- handle authentication using passwords
- use FTP to download files
- use persistent connections
- upload files
- handle escaping characters in requests
- handle binary data

High-Level Functions in RCurl

High-level functions

There are 3 high-level functions in `"RCurl"`:

Function	Description
`getURL()`	fetches the content of a URI
`getForm()`	submits a Web form via the `GET` method
`postForm()`	submits a Web form via the `POST` method

High-Level Functions in RCurl

Main Characteristics of high-level functions

- they take a URI and other optional parameters
- they send an HTTP request and expect a document in response
- they differ in the type of document they retrieve and how

Function getURL()

As its name indicates, this function allows us to fetch the contents of a URL.

`getURL()` expands the somewhat limited capabilities provided by the built-in functions `download.url()` and `url()`

Function getURL()

Using getURL()

getURL() downloads *static or fixed content* files. It collects and returns the body of the response into a single string.

For instance, let's fetch the content of the R-project's homepage

```
# load RCurl (remember to install it first)
library(RCurl)

# retrieving the content of the R homepage
Rproj = getURL("http://www.r-project.org/")
```

Function getURL()

If you inspect the content of Rproj, you should see this:

```
[1] "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">\n
<html>\n<head>\n<title>The R Project for Statistical Computing</title>\n
<link rel=\"icon\" href=\"favicon.ico\" type=\"image/x-icon\">\n<link rel
=\"shortcut icon\" href=\"favicon.ico\" type=\"image/x-icon\">\n<link rel
=\"stylesheet\" type=\"text/css\" href=\"R.css\">\n</head>\n\n<FRAMESET cols
=\"1*, 4*\" border=0>\n<FRAMESET rows=\"120, 1*\">\n<FRAME src=\"logo.html\"
name=\"logo\" frameborder=0>\n<FRAME src=\"navbar.html\" name=\"contents\"
frameborder=0>\n</FRAMESET>\n<FRAME src=\"main.shtml\" name=\"banner\"
frameborder=0>\n<noframes>\n<h1>The R Project for Statistical Computing</h1
>\n\nYour browser seems not to support frames,\nhere is the <A href=\"navbar
.html\">contents page</A> of the R Project's\nwebsite.\n</noframes>\n
</FRAMESET>\n\n\n\n"
```


Function getURL()

getURL() output

What can we do with the retrieved content in Rproj?
We can parse it using the functions from the "XML" package:

```
# load XML
library(XML)

# parsing the content of the R homepage
Rproj_doc = htmlParse(Rproj)
```

```
## Loading required package:  methods
```

RCurl for web Forms?

"RCurl" and Web Forms

The real power and *raison d'etre* of "RCurl" is its capacity for making requests associated to Web Forms

"RCurl" for Web Forms

Go check the next slides!!!

