RCurl - an R package for HTTP requests

Duncan Temple Lang
Department of Statistics,
Universy of California at Davis.

February 7, 2012

Abstract

The Hyper-Text Transfer Protocol (HTTP) is the communication underlying much of the Web and Web services, allowing Web browsers to download pages, submit HTML forms, and other applications to perform rich distributed computing via Web Services such as SOAP. The RCurl package provides an interface to a well-established, general and flexible C library – libcurl – that performs HTTP requests. We describe the primary functions in the R package and the basic computational model exposed via this interface. We describe some of the more advanced and flexible features of the package and provide some examples. We contrast it with other potential HTTP client packages for R and suggest some potential improvements that could be made in the package.

1 Introduction

In section 3, we provide an overview of the Hyper-Text Transfer Protocol. This helps to illustrate the large number of facilities one must implement to provide a complete HTTP client facility. In section

2 ??

RCurlBasics, we present the primary functionality of the RCurl package and illustrate these with some basic examples. In section 5, we describe some of the more advanced features of the package that can be used to provide more controlled access to the HTTP requests and the responses. In the final section, we discuss some alternative approaches to providing client-side HTTP facilities in R and some potential improvements to the RCurl package. The goal of the paper is to provide a flavor for what the RCurl package can do and how to think about it when using it to develop Web-based applications in R. The paper is not intended to be a complete programmers reference guide. One should consult the help pages for both RCurl and libcurl for detailed information about the possible options one can use.

3 The Hyper-Text Transfer Protocol

Almost everyone using the Web in some form will be familiar with the term 'http' that is the most common prefix for web sites. This stands for the Hyper-Text Transfer Protocol and is represents a language or protocol by which a Web client and server can converse. Defined by a W3 document [**?**], it specifies how a client can initiate a conversation with a Web server, how the two can negotiate the settings each will use, how the client can request specific documents or content and how this material is to be transferred. Ths simplest use is the most common and involves downloading a file from a Web server. Let's consider the essentials of what happens when we download the URI (Uniform Resource Identifier) using a Web browser.

Redirects

Request Body uploads using boundary strings

Errors

Suppose we were to download the file index.html from the RCurl web page, i.e. http://www.omegahat.org/RCurl/index.html. Our client would use TCP/IP to connect to the machine www.omegahat.org (having first resolved the IP address of the machine corresponding the name www.omegahat.org). It would attempt to connect on the port 80 which is the default port on which Web servers listen for such requests. If the server were listening on a non-standard port, such as 8080, the request would be http://www.omegahat.org:8080/RCurl/index.html and the connection would be established using that different port. Assuming the connectin has been made, the client and server then start their HTTP conversation. The client writes the basic command to fetch the file /RCurl/index.html:

```
GET /RCurl/index.html HTTP/1.1
```

The word GET tells the server what operation is being requested, in this case the retrieval of a document. The next part of the command identifies the document. And the final part tells the Web server that the client wishes to communicate using HTTP rather than any other protocol, and specifically version 1.1 of the protocol rather than 1.0. This more recent version of the protocol provides enhanced facilities for the client and server to provide information controlling the connection between them.

Because we are using the 1.1 protocol, the client must provide information identifying itself the application in addition to this basic retrieval instruction. Specifically, it must identify the domain of the Web server from which it is requesting the document. This might seem strange; surely the Web server knows the domain of the Web server itself. This is not always the case as a single Web server can act as a virtual server for many domains. And so, in version 1.1 of HTTP, the client should indicate the domain of interest.

So the client need only write the following text along the socket connecting the client and Web server:

```
GET / HTTP/1.1
Host: www.omegahat.org
```

Both lines are ended with a control-linefeed (

r

n) and the request is completed by sending a terminating blank-line.

The Web server is reading the bytes on the socket and then recognizes the end of the request. It then processes the request and provides a response. Each reply has the same basic form. The first line provides information about the request. The last component provides an indication about the status of the request. The numbers indicate success or failure adn identify particular reasons for that response. The next segment of the response is a collection of name–value pairs given as

```
name: value
```

These provide information for the client telling it how the information in the response is encoded. This segment is terminated by a blank line and then the data for the response appears is written to the connection. The client can then read this information

Figure 1: The anatomy of an HTTP response

4 RCurl and libcurl

4.1 The Basic Computational Model

This describes the basic functions of requesting a URI and submitting a form. It describes how we can have a handle that we can reuse across multiple requests or create one each time. It discusses the use of options via the . . . mechanism to set libcurl options within a call or persistently within a libcurl handle that apply to all requests using that handle.

4.2 Downloading a URI.

4.3 Forms

Submitting a form using GET and POST.
Google. basic CGI env. wormbase
Discovering the Available Options

4.4 libcurl and default functionality

4.4.1 .netrc

4.4.2 SSL

5 Advanced Features

5.1 Accessing the header information

5.2 Uploading a File

6 Alternative Approaches and Future Work

httpRequest not very complete.
Other libraries such as libwww. libcurl has many features and is very portable.
Event loop.
multi-threaded libcurl.