The `websockets` Package

Bryan W. Lewis
blewis@illposed.net

May 6, 2011

1 Introduction

The `websockets` package is an HTML 5 Websocket implementation for the R language, based on the C `libwebsockets` library written by Andy Green. The `websockets` package is especially well-suited to lightweight interaction between R and web scripting languages like Javascript. Multiple simultaneous websocket connections are supported.

By "lightweight" we mean that the library has few external dependencies and is easily portable. More significantly, `websockets` lets Javascript and other scripts embedded in web pages directly interact with R, bypassing traditional middleware layers like .NET, Java, and web servers normally used for such interaction.

The HTML 5 Websocket API is a modern socket-like communication protocol for the web. Note that the HTML 5 Websocket API is still under development and may change. Some browsers may not enable Websockets by default (including recent versions of Firefox), but there are usually simple methods to enable the API. Despite its developmental status, the API is presently widely supported: most recent browsers support it and there are many available langauge implementations.

2 Using `websockets`, step by step

The `websockets` library may be viewed as a server that can initiate and respond to HTML 5 websocket and HTTP events over a network connection (Websockets are an extension of standard HTTP). The library is intentionally somewhat simple and tries to implement most important functions of the interface in the R language.

All R/Websocket applications use the following basic recpie:

1. Load the library.

2. Initialize a websocket context with `createContext`.
3. Set callback functions that will respond to desired events.
4. Accept requests from connections (e.g., Javascript code).
5. Service the socket interface with `service`, often in an event loop.
6. Delete the context environment when done.

We outline the steps below.

2.1 Load the library

```
library('websockets')
```

The `websockets` library suggests the `RJSONIO` library be installed, as it is quite useful to have available when interacting with Javascript.

2.2 Initialize a websocket context with `createContext`

The R/Websocket service is initialized by a call to the `createContext` function. The function takes 2 arguments, a network port to listen on, and an optional file name of a file to service standard HTTP requests. The function returns an environment, into which so-called "callback" functions may be assigned that will respond to websocket events. Here is an example:

```
context = createContext()
```

HTML 5 Websockets may be be used directly by Javascript embedded in arbitrary web pages. The `websockets` library can respond to websocket requests on its port that are not associated with any local HTML web page. For an example of this, see the package demo available from `demo('json.R')`.

For convenience, the `websockets` library includes the ability to serve an HTML file to incoming HTTP requests on its port. For example, the basic package demo available from `demo('websockets')` serves clients the file `basic.html` located in the package installation path. However, serving HTML web pages is not the primary function of the `websockets` library and there are many other excellent alternatives for that available to R.

2.3 Set callback functions to respond to events

Clients may connect to the websocket service immediately after the context is initialized. Nothing interesting will happen however until callback functions are defined to respond to events.

The `websockets` package presently supports the following events:

`established`: Occurs when a websocket connection is successfully negotiated.

`closed`: Occurs when a client closes its websocket connection.

`receive`: Occurs when data is received from a connection.

`broadcast`: Occurs when data is received from a broadcast event.

R functions may be defined to handle some, all, or none of the above event types. Such functions are termed "callbacks."

The `setCallback` function may be used to define a callback function in the websocket context environment returned by `createContext`. (It simply assigns the functions in that environment.) A callback function must take precisely 3 arguments that are filled in by the library with values corresponding to an event when invoking a callback function. The values are:

`DATA`: A RAW vector that holds any incoming data associated with the event. It may be of length zero if the event does not have any data to report.

`WS`: A pointer reference to the websocket connection associated with the event.

`COOKIE`: A pointer to a data 'cookie' associated with the websocket client connection.

The arguments can be named arbitrarily, but there must be three.

The example function below assigns a random number (in character form) to the cookie of a new client connection after a connection is established, and then sends a message to the connection:

```
f = function(DATA, WS, COOKIE) {
  x = runif(1)
  setCookie(COOKIE, paste(x))
  websocket_write(paste("Connection established. Your cookie value is",x), WS)
}
setCallback("established", f, context)
```

Here is an example function callback that receives data from a client connection and simply echoes it back:

```
f = function(DATA, WS, COOKIE) {
  websocket_write(DATA, WS)
}
setCallback("receive", f, context)
```

2.4 Accept requests from web clients

Javascript and other web script clients can very easily interact with the R `websockets` library directly from most browsers. The listing below presents a very basic javascript example, see the `basic.html` file in the package installation path, or the `http://illposed.net/rwebsockjson.html` file for more complete examples.

```
<html><body>
<script>
socket = new WebSocket("ws://localhost:7681", "R");
try {
  socket.onmessage = function got_packet(msg) {
    document.getElementById("output").textContent = msg.data;
  }
catch(ex) {document.getElementById("output").textContent = "Error: " + ex;}
</script>
<div id="output"> SOCKET DATA APPEARS HERE </div>
</body></html>
```

Note, in particular, that the `websockets` package defines a single protocol called "R." Future versions of the package will admit multiple protocols.

2.5 Service the socket interface with `service`

Websocket events are placed in a queue. The `service` function processes events in the queue on a first-come, first-served basis. The `service` function processes each event by invoking the appropriate callback function. It returns without blocking if there are no events to service. Events may be processed indefinitely by evaluating the `service` function in a loop, for example:

```
while(TRUE)
{
  service(context)
  Sys.sleep(0.05)
}
```

Although this is a polling loop, the `Sys.sleep` function prevents the R session from spinning and consuming lots of CPU time. A blocking version of `service` is not presently available, but this simple approach works surprisingly well.

2.6 Delete the context environment when done

Underlying low-level data pointers used by the library are automatically de-allocated when the environment returned by `createContext` is deleted, and the garbage collector has run.

```
rm(context)
gc()
```

A new context may be created any time after this point. We are conisdering forcing the garbage collector to run automatically when a context is deleted, but are not sure about the implications of doing that yet.

3 Tricks and miscellaneous notes

We present a few more advanced and other miscellaneous notes in this section.

3.1 Binary data

It is possible to exchange binary data over websockets, and the default I/O type on the R side of the library is the R `raw` type. For convenience, R character variables are cast to `raw` by the library automatically, but all other types require manual serialization to `raw` prior to transmission.

JSON is probably a good choice to use when interacting with Javascript and the data size is not too large. The suggested `RJSONIO` package helps map many native R objects to JSON and vice versa, greatly facilitating interaction between R and Javascript. But, JSON data is transferred as characters, which may incur performance and in some cases numeric issues.

3.2 Broadcasting

The HTML 5 Websocket API is a peer to peer, connection oriented protocol and does not specifically include a way to broadcast data across multiple connections.

The `websockets` library emulates broadcasting with a trick, the `websocket_broadcast` function. The `websocket_broadcast` function takes a single DATA argument (without specifying a websocket connection). When invoked, it induces the `broadcast` callback with the specified DATA payload once for each connected client websocket on the server. The `broadcast` callback may then use `websocket_write` to send out the DATA payload to each client websocket in succession.

Using this trick, the `websockets` library can quickly send the same DATA to all connected websockets.

3.3 Pushing data to a connection

The `websocket_write` function may be called at any time to write data to a specific websocket connection. However, `websocket_write` requires a pointer to the connection which is usually only available from inside a callback function.

There are at least two viable approaches to writing data to a connection from the server to a client without requiring the client initiate the transfer.

1. Maintain a list of active client websocket connections by adding to the list in the `establish` function callback, and removing from the list in the `close` function callback. This will further require that the connections are uniquely identifiable by setting an appropriate data cookie. In this way, `websocket_write` may be called at any time to send data to connetions in the active list.
2. Use the `websocket_broadcast` function as outlined in the last section.

The `websocket_broadcast` function is simple, but writes the same data to all the clients. Use the other approach if you need to push data to multiple clients.

4 Interacting effectively with Javascript

Javascript has excellent HTML 5 Websocket support. The suggested `RJSONIO` package provides simple methods for exchanging data between R and Javascript using JSON, suitable for smallish data sizes.

The Javascript available in the HTML page `http://illposed.net/rwebsockjson.html` and the R code available from the `demo('json')` demo illustrate using `websockets`, JSON and flotr to implement basic dynamic R web plots.

5 Up next...

We've left out lots of the Websocket API features. Some of them that we plan on implementing next include:

- SSL encrypted communication.
- Multiple simultaneous protocol support.
- A blocking `service` function?
- Smarter object de-allocation and clean up