WebDevelopR: A Website Development Package for R

Evan Ray
University of Massachusetts, Amherst

Peter Krafft
Massachusetts Institute of Technology

John Staudenmayer
University of Massachusetts, Amherst

Abstract

Many statistical methods would be useful to non-statisticians who lack the ability to program the method in a language such as R or even use programs written by others. One way to address this is for statisticians to create websites that receive data, automatically run the statistical method, and produce output. The purpose of this paper is to introduce an R package and tools that make this relatively easy to do. Our work augments and improves upon the R package **CGIwithR**. We review that and several other packages and discuss their benefits and drawbacks. We then present the **WebDevelopR** package, which addresses some of those drawbacks. We discuss two versions of this new package. The first is relatively simple and is intended for statisticians without experience in website development, and the second is intended for statisticians who have more experience in creating websites. Both versions are illustrated with simple examples.

Keywords: AJAX, CGI, HTML, JavaScript, Perl.

1. Introduction

It is common for statisticians to develop methods to process and analyze data stored in formatted files, such as output from electronic devices or laboratory instruments. Many statisticians then make those methods available to others through an R (R Core Team 2012) package or other software. However, using these software packages can be difficult for non-statisticians who want to apply these methods to their data. One option for making this easier is to build a website that serves as a graphical user interface for the statistical analysis software.

This paper describes an R package, **WebDevelopR**, and a set of tools to create such a website. These new tools address some technical difficulties with the earlier package **CGIwithR** (Firth 2003). They also aim to fill a gap in existing solutions by providing a method of building a website to run R scripts that makes it comparatively easy to both create and customize the website. There are two versions of the **WebDevelopR** package. One version, `turnkey`, is simpler to use and is intended for a statistician with little or no experience in website design. It allows the developer to create a simple website with a form for the site user to provide input to the R script and a page to display the results of the analysis. This requires only

knowledge of R and enough familiarity with HTML to edit a template form to provide the desired user input fields. The second version of the package, `developer`, is intended for the statistician who is comfortable with JavaScript and HTML, and it is more flexible.

The remainder of the paper is organized as follows. In Section 2 we describe several existing approaches, their benefits, and their limitations. We focus on the package **CGIwithR**, and discuss some technical problems a developer might encounter in using that package. In Section 3 we present a new R package, **WebDevelopR**, that addresses those drawbacks and technical difficulties. We describe two versions of this package and demonstrate how each can be used through an example website. In Section 4 we discuss some security considerations to be aware of when using **WebDevelopR**. The paper concludes with a brief discussion. A separate file with appendices contains the code used in the examples, detailed instructions describing how to use the package, documentation of R utilities in the package, and a list of resources we have found useful to learn about web development. A reader who simply wants to use the package can skip to Section 3.

2. Existing approaches and technical background

The ability to use R for statistical analysis inside of a website application is widely useful, and there are many tools available that make this possible. Each of these tools has different purposes, strengths and weaknesses that make it more or less appropriate for a particular project. We briefly discuss five current options to give a sense of what is available, and then we describe a sixth tool, **CGIwithR**, in more detail. It is the basis for our approach. For more tools and examples, two resources about web-based R are

- `http://cran.r-project.org/doc/FAQ/R-FAQ.html#R-Web-Interfaces`, and
- `http://biostat.mc.vanderbilt.edu/twiki/bin/view/Main/StatCompCourse`

Table 1 provides the current web address of each approach we discuss.

One of the simplest choices for web development using R is **Rweb**, which provides a template for a website that has an embedded R application. Three versions of the website are described in Banfield (1999). The first is a command prompt that is initialized to hold whatever code the developer wants to present to the user. The user may delete or otherwise edit the code in the prompt. The second is a similar interface that uses JavaScript to enhance the appearance of the site. The third provides an interface to run a few pre-selected R routines. The main purpose of these tools is for running low intensity scripts on the host servers, such as class homework assignments. The websites provided by **Rweb** can be used as templates for developing new sites, but knowledge of Perl and HTML is required to do that.

Rwui (Newton and Wernisch 2007) is another tool for rapidly developing a website application that runs an R script. The **Rwui** website uses a questionnaire to learn what user inputs, analyses, and output the developer would like to include. It then automatically builds code for a website that runs with the Apache Tomcat web server. The generation process is quick and requires practically no knowledge of web development, but the resulting website cannot be easily customized.

FastRWeb (Urbanek 2011) is an option that interfaces with R through the **Rserve** (Urbanek 2012) package. The advantage of this is that an instance of R is initialized before the page

Utility Name	Current URL
Rweb	`http://www.math.montana.edu/Rweb/`
Rwui	`http://sysbio.mrc-bsu.cam.ac.uk/Rwui/`
FastRWeb	`http://www.rforge.net/FastRWeb/`
rApache	`http://biostat.mc.vanderbilt.edu/rapache/`
shiny	`http://www.rstudio.com/shiny/`
CGIwithR	`http://www.omegahat.org/CGIwithR/`

Table 1: Current URLs of the tools discussed.

request is received by the web server; this enables rapid response times. Creation of a fully funtional web application using **FastRWeb** requires knowledge of HTML and JavaScript though.

The goal of **rApache** (Horner 2013) is to be able to write a website in R without using a scripting language such as Ruby or PHP. It consists of an Apache module for linking to R and an R package for linking to Apache. **rApache** provides a number of functions that allow web development to be totally controlled by R. That large degree of flexibility comes at the cost of being substantially more difficult to use. **rApache** benefits from a large number of online examples and tutorials.

A more recent option is **shiny** (RStudio, Inc. 2013). This package makes it very easy to build websites that interact with R without any knowledge of web programming or other scripting languages. A drawback to **shiny** is that applications developed with it can only be deployed online with the **shiny** web server, which has only been created recently and has a relatively limited feature set. Integration with more feature-rich web servers such as Apache will not be feasible without more development.

CGIwithR (Firth, 2003) is similar to several of the above packages in that it enables the developer to use R as a CGI scripting language (i.e., to be able to dynamically generate content to be displayed in a web browser, possibly using input from a web form in order to do so). It accomplishes this by providing a Perl script that acts as a connecting layer between the web server and R. When a web form is submitted to the web server, the server passes the submitted data on to this Perl script. The script stores the submitted data in environment variables and then runs the desired R script. When R loads, the **CGIwithR** package pulls any submitted information from the environment variables and stores it in a native R object. The R script then processes the data, and any generated output is captured. When R exits, the Perl script sends this captured output to the web browser where it is (typically) displayed to the site user as a new web page. This process is illustrated in Figure 1.

The **CGIwithR** package is often a good solution, but it has limitations when statistical analysis requires the user to upload large data files or when the analysis takes an extended time to complete. We discuss the causes of these problems and what **WebDevelopR** does to address them here; additional features of **WebDevelopR** are discussed in Section 3.

One problem encountered with **CGIwithR** is that a limited amount of data can be stored in the environment variables used to pass submitted data to R. The exact size limits depend upon the configuration of the server, but the result is that it can be impossible to upload large data files. On our web server, we were unable to upload files that were about 6MB. In

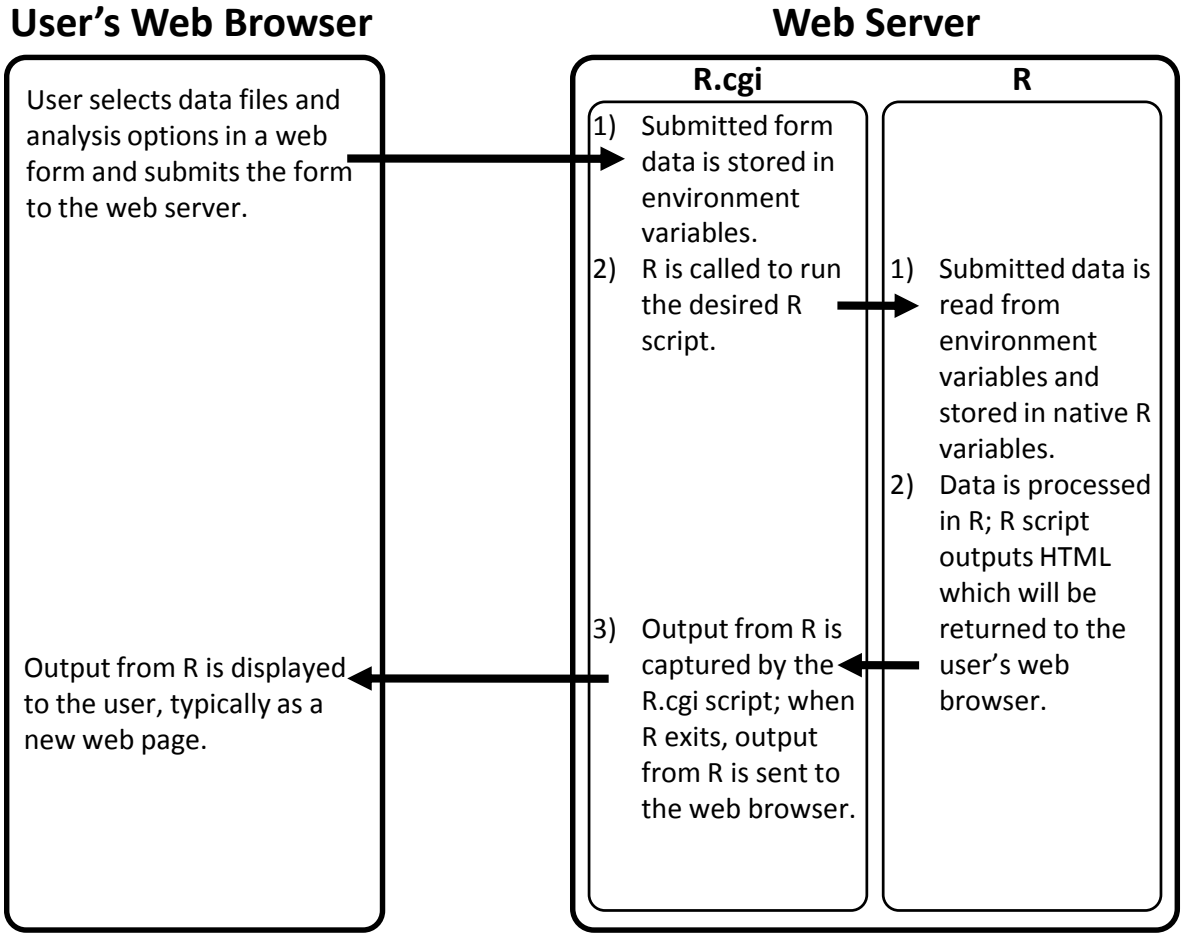

Figure 1: Diagram of a server call using **CGIwithR**.

order to solve this problem, **WebDevelopR** stores submitted data in temporary files on the hard drive, rather than in environment variables. Each time the script is called, a unique session ID is generated to identify these temporary files, and the locations of these files are passed to R via command line arguments.

A second problem with **CGIwithR** is that the connection between the web server and the site user's browser may time out while the R script is still running. When this happens, no results are sent to the user and the web server typically terminates any CGI scripts associated with the connection. Most web servers provide configuration options to set the timeout length (for example, in the Apache HTTP Server the `TimeOut` directive can be used). However, even if this setting is changed on the server, the user's browser may terminate a connection after a few minutes of inactivity (see for example the `network.http.keep-alive.timeout` setting in the Mozilla Firefox browser). **WebDevelopR** resolves this problem by periodically sending a character of data to the user's browser while the R script is executing. These characters are not displayed to the user, but they keep the connection alive until the analysis is complete.

Creation of websites with **CGIwithR** requires knowledge of HTML and JavaScript. The `turnkey` version of the **WebDevelopR** package alleviates this requirement by providing a template website that requires only minor edits to the HTML to create a useful site. This template website includes some user interface features tailored for lengthy statistical calculations, using AJAX (Asynchronous JavaScript and XML) to provide status updates to the website user while the calculations are progressing. Similar functionality could be created with **CGIwithR** or several of the other packages we have discussed, but it would require knowledge of JavaScript. On the other hand, the `turnkey` version of **WebDevelopR** provides a simple R function to return status updates to the website user. The `developer` version of the package also facilitates development with AJAX by allowing the developer to easily specify the data type of the response from the CGI script. These features are discussed more in Section 3.

3. New approach

Two versions of the **WebDevelopR** package are available, `turnkey` and `developer`. Both versions of the package use the same general framework as **CGIwithR**: Perl scripts provide an interface between the web server and R. The main differences between **CGIwithR** and the **WebDevelopR** lie in the functionality provided by that interface. Several of these new features were mentioned in the previous section: submitted form data is stored in temporary files on the hard drive rather than in environment variables; the connection between the web browser and the server is kept alive during lengthy computations by sending a character of data to the browser every second; and some features are added to simplify the use of AJAX.

The `turnkey` version of the package builds on that foundation and provides a template web application consisting of a form and a results page, and a set of R functions to facilitate website creation with only basic knowledge of HTML. This template can be set up to run a desired R script by editing configuration settings and modifying the provided web form to include the necessary user input fields; no knowledge of languages like JavaScript or Perl is required. The `developer` version provides only the basic functionality for passing data from submitted web forms to R, keeping the connection between the server and the web browser alive during lengthy computations, and returning the output from R to the web browser. This version is intended for experienced web programmers who want to make use of the opportunities for rich

user interactions made possible by AJAX. We describe each of these versions of the package in detail and provide examples in Sections 3.1 and 3.2.

3.1. Turnkey version: For people with little experience as web programmers

In this section we describe the `turnkey` version of the package. We give an overview of how it works, discuss package use and installation, and give an example.

Overview

The `turnkey` version of **WebDevelopR** combines several HTML, JavaScript, and Perl files with some R code to create a simple web application. The web site user's primary point of interaction with the application is a web form. This form is configured by the application developer, and allows the user to select data files to upload to the server for analysis and/or choose other options for the analysis. The form submission is processed by an R script on the server, and results are displayed on a new web page.

When the user submits the form, the application follows one of two paths depending on whether the user's web browser has JavaScript enabled. If JavaScript is not available, the form is submitted to the `processForm.cgi` script, which is written in Perl. This script does the following:

1. The script generates a unique session ID which is used throughout processing to keep track of temporary files and options the user selected. Session IDs are assigned sequentially and are obtained from a plain text file on the web server. Please see the security notes in Section 4 below for further discussion.

2. It then stores the data from the submitted form in temporary files. These temporary files will be accessed later by the R script.

3. It sets up a temporary file to store status updates for the user. Users who do not have JavaScript enabled will not see these status updates during processing, but they will be displayed when processing is complete.

4. The script next creates a hidden HTML element and a thread that prints a '.' to the browser every second in order to keep the connection from timing out. This thread will continue to run until R has finished running.

5. The script now runs an R script, which loads the submitted data from the temporary files, processes it, and outputs results to be displayed to the user in HTML. R code is provided to read the data from temporary files and store it in native R variables.

6. When the processing in R is completed, the thread printing periods is killed, status updates are retrieved and returned to the user's browser, and output from R is returned to the user.

7. Finally, all temporary files created in steps 2 and 3 above are deleted.

This process is illustrated in Figure 2.

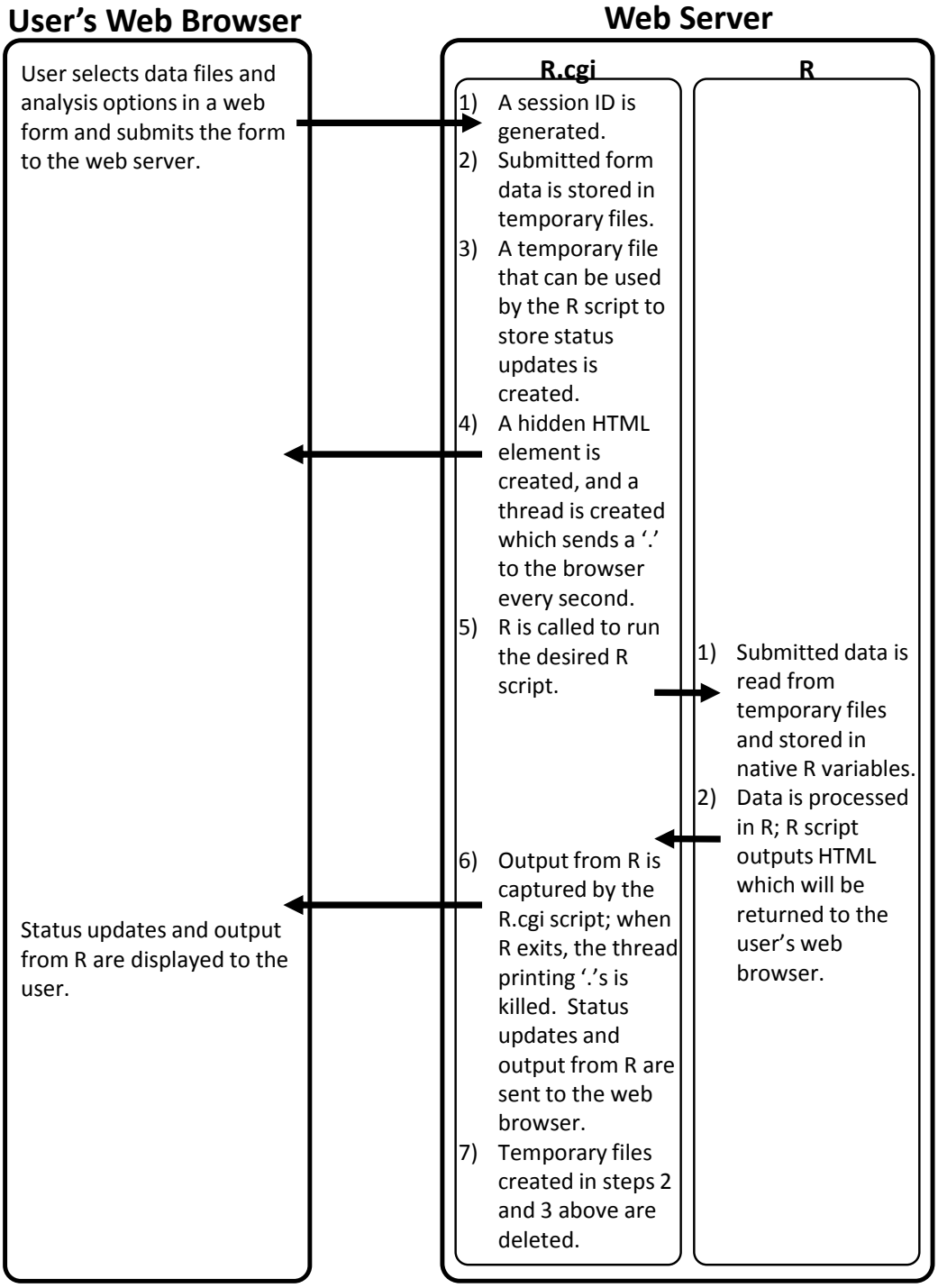

Figure 2: Diagram of a server call using **WebDevelopR** when JavaScript is not available on the user's web browser.

If JavaScript is available, the functionality in `processForm.cgi` is divided into several stages to provide a better user experience. In this case, when the form is submitted it is sent to the `preProcessForm.cgi` script, which handles items 1 and 2 of the `processForm.cgi` script. Once the form preprocessing is completed, the user's browser loads the results page. From the results page, two different scripts on the web server are called. First, the wrapper script `R.cgi` runs. This script handles the remaining items 3 through 7 in the list above. The results page also periodically calls `statusUpdate.cgi`, which retrieves any status updates provided by the R script and sends them back to the user's browser where they are displayed when they are received.

Package Use and Installation

When R loads, it has access to the submitted form data through two functions: `form.data` and `file.details`. The `form.data` function returns a list with one component for each field in the submitted form. The names of the components in this list are taken from the `name` attribute of the corresponding form fields in the HTML document. If the name of the form field in the HTML document began with a number, it is prepended with an X. In the case of file upload fields, the corresponding entry in the list contains the full path to the temporary file on the server where the uploaded file was stored (or an empty string if no file was uploaded). For other form fields, the corresponding component of the list is a vector of submitted values (or an empty string if no options were specified for that form element). For example, for a group of checkboxes the entry in the list will be a vector of the values of the checkboxes that were selected by the site user. The list returned by `form.data` also includes one other component: `sessionID`, which contains the session ID assigned to the form submission.

Additional information about each uploaded file are in the list returned by `file.details`, which only contains entries for form fields where a file was selected. For each uploaded file, this list contains a vector of three values:

1. `file.name` is the original file name of the uploaded file (including the extension),
2. `content.type` is the MIME content type of the file, and
3. `text.or.binary` is Perl's educated guess as to whether the uploaded file is a text or binary file. Either "text" or "binary".

WebDevelopR includes several utility functions that can be used to facilitate web development. These are summarized in Table 2, their use is demonstrated in the sample code in Appendix A, and more complete documentation is given in Appendix D. See also Appendix E for some other resources and R packages that can be helpful in web programming.

There are seven major tasks to do in order to set up a working website with the `turnkey` version of **WebDevelopR**, which we describe next. More detailed information about each of these steps is in Appendix C.

1. Install a Perl interpreter and necessary modules. Perl is included by default on most Unix-like operating systems. For Windows, you'll need to install an implementation of Perl such as Strawberry Perl (`http://strawberryperl.com/`). You will also need to ensure that the following Perl Modules are installed (typically, they will be by default):

Function Name	Description
`status.update`	Provides status updates to the user while the R script is still running.
`web.jpg`, `web.png`	Used to create .png and .jpeg image files, and to embed these images in the results page.
`web.print`	Prints R objects in a formatted `<pre> ... </pre>` HTML environment (similar to verbatim in LaTeX).
`web.table`, `web.csv`, `web.csv2`	Prints an R data frame or matrix to a text file and inserts a link to the file in the results page.

Table 2: The utility functions included in **WebDevelopR**.

- `CGI`
- `Fcntl`
- `HTML::Entities`
- `Time::HiRes`
- `threads`

2. Install and configure a web server. **WebDevelopR** can be used with any web server that supports CGI scripts written in Perl. The Apache HTTP server is one common choice (`http://httpd.apache.org/`). It should be configured to enable CGI scripts.

3. Install the **WebDevelopR** package. Note that if your webserver will run with a different username than yours, you may need to install the package to a site library to ensure that an instance of R created by your web server will have access to the package. This can be achieved with the `lib` argument to the `install.packages` command in R.

4. Copy files included with **WebDevelopR** to appropriate locations on your server. Copy the files in the `turnkey\cgi-bin` folder in the installation directory to your web server's `cgi-bin` folder, and the files in the `turnkey\html` folder in the installation directory to your web server's `html` folder (or equivalent – for the Apache server on Windows, this folder is named `htdocs`). Permissions on these files should be set so that the web server can execute the CGI scripts and read the files in the `html` folder. Also create a `system` directory outside of your web server's `html` folder and the `cgi-bin` where "system" files can be stored and copy the contents of the `turnkey\system` folder in the installation directory to it. Your web server should have read and write access to the files in this directory.

5. Edit configuration settings in CGI and JavaScript files. Each of the CGI scripts in the `cgi-bin` and the JavaScript scripts in the `html\scripts` folder has a section at the top with configuration settings for things like the locations of files on the web server, the title at the top of the web page, and how frequently status updates are retrieved.

6. Modify the provided `index.html` file to provide form fields for site users to enter input. The file includes a form with examples of all of the basic form input field types.

7. Modify your R script to use submitted form data and provide output to site users and place it in the web server's `cgi-bin` folder. See the example code in Appendix A and the package documentation for more information.

Example

In this section we present a simple example of how the `turnkey` version of **WebDevelopR** could be used. Our example is not very exciting, but it demonstrates the type of functionality that can be achieved. The R code used for this example is in Appendix A. This R code uses the utility functions provided by **WebDevelopR** (listed in Table 2) to send status updates to the website user and include printed R objects, graphics, and links to .csv files on the results page.

The form pictured in Figure 3 allows the site user to select a data file to upload, enter a name and description of the analysis to be displayed in the results, choose which summary statistics will be calculated, and select the image format for a plot of the data.

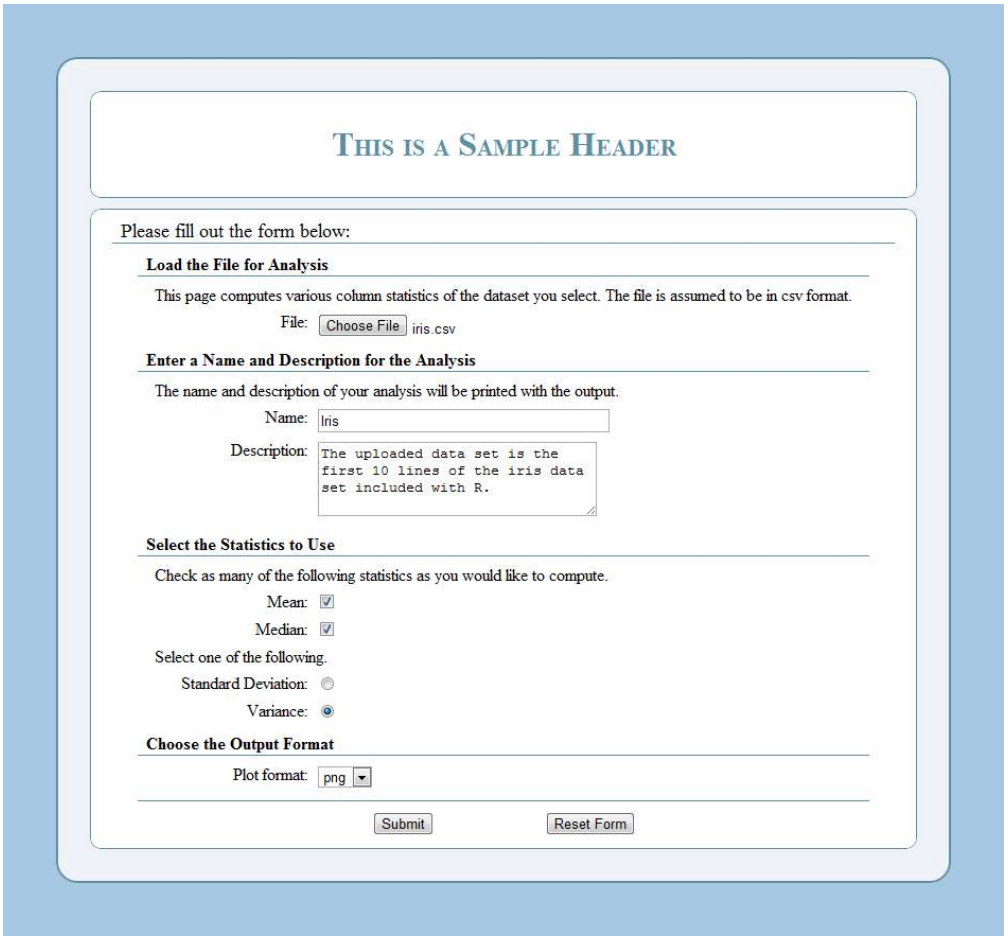

Figure 3: Screenshot of the web form for the `turnkey` version

Once the user has entered this information and clicks the submit button, she is taken to a waiting page while the R script runs. While on this page, status updates provided by the R

script are retrieved from the server and displayed to the user every few seconds. This screen is pictured in Figure 4.

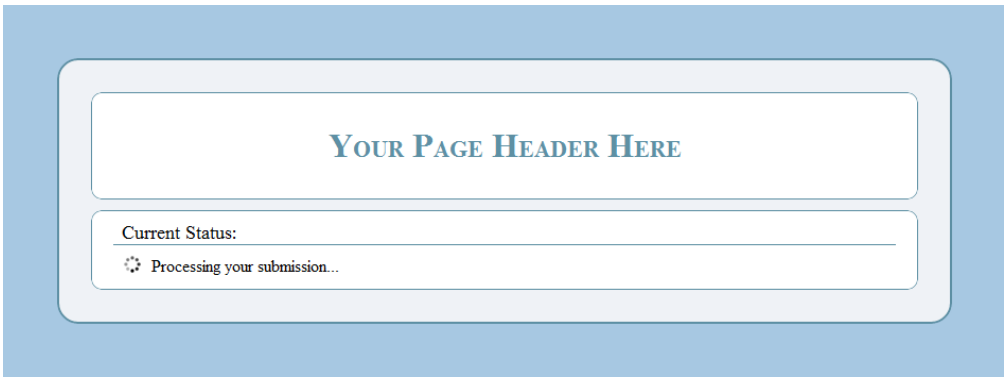

Figure 4: Screenshot of the waiting page for the `turnkey` version

When the analysis is complete, all status updates are displayed to the user as well as the results of the analysis (a print of the data set, a plot, and a link to a .csv file containing more results). A screenshot of this page is in Figure 5.

3.2. Developer version: For more experienced web programmers

We now turn to the `developer` version of **WebDevelopR**. Again, we describe how it works, discuss package use and installation, and give an example.

Overview

The `developer` version of **WebDevelopR** provides a Perl script that processes form submissions, runs an R script, and sends the output from R to the website user. The provided Perl script does the actions below. Note that some of these are the same as in Section 3.1. We list them again here to make the paper easier to read.

1. The script generates a unique session ID which is used throughout processing to keep track of temporary files and options the user selected. Session IDs are assigned sequentially and are obtained from a text file on the web server. Please see the security notes in Section 4 below for further discussion.

2. It then stores the data from the submitted form in temporary files. These temporary files will be accessed later by the R script.

3. The script next creates a JSON, XML, or hidden HTML element (depending on the response document type, which can be specified via a configuration setting). It then creates a thread that prints a '.' to the browser every second in order to keep the connection from timing out. This thread will continue to run until R has finished running.

4. The script now runs the desired R script, which loads the submitted data from the temporary files, processes it, and outputs results to be displayed to the user in JSON, XML, or HTML. R code is provided to handle reading the data from temporary files and

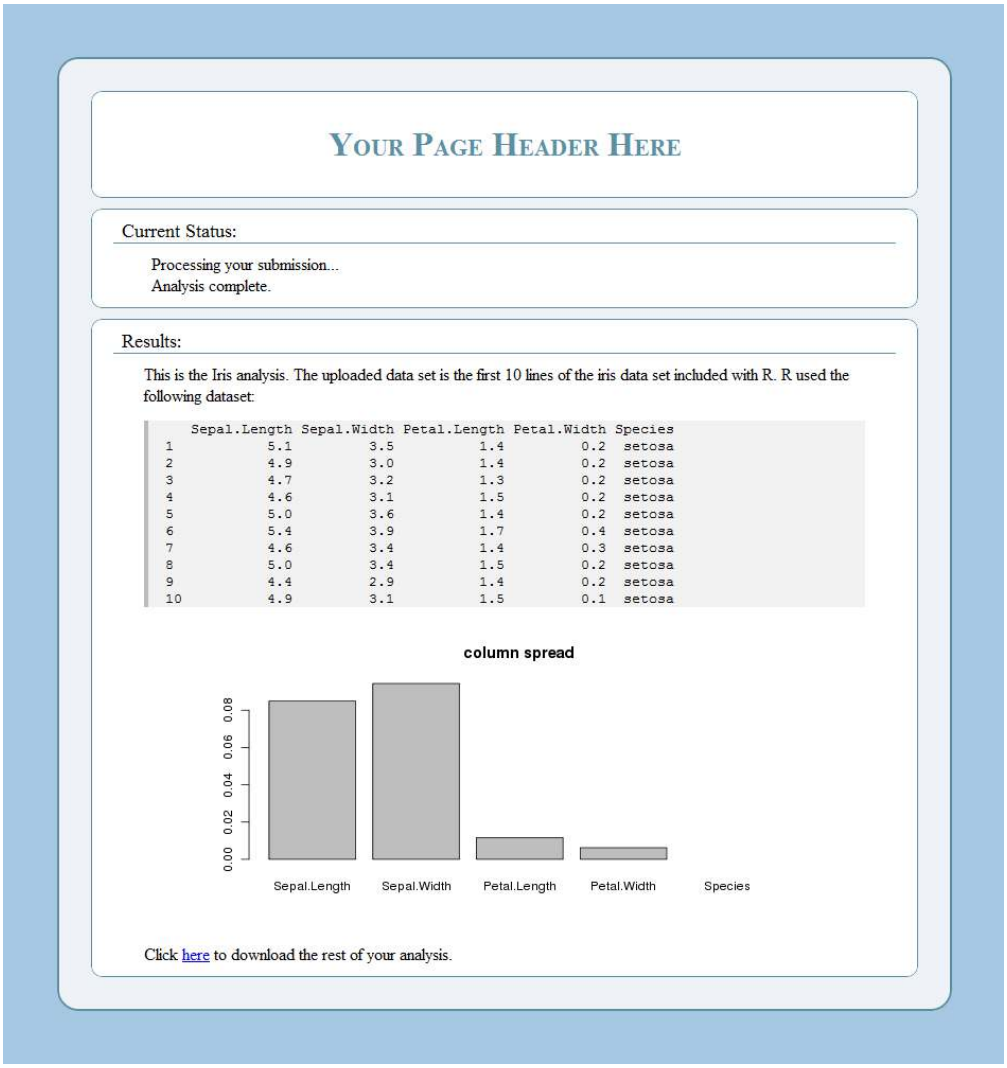

Figure 5: Screenshot of the results page for the `turnkey` version

storing it in native R variables. A hidden input element in the submitted form specifies the name of the R script to call; for security reasons, this script name is validated against a list of scripts specified in the CGI script that processes the form submission.

5. When the processing in R is completed, the thread printing periods is killed and the output from R is returned to the user's web browser.

6. Finally, all temporary files created in steps 2 and 3 above are deleted.

Package Use and Installation

The `form.data` and `file.details` functions can be used to access the uploaded data as described in Section 3.1.1 above. The same steps are required to install the `developer` version as the `turnkey` version with a few small changes. In step 4, the files should be copied from

the `developer\cgi-bin` and `developer\system` folders in the package installation directory instead of the corresponding `turnkey` folders. When using the `developer` version of the package all HTML, CSS, JavaScript, and other desired CGI functionality must be programmed by the site developer. Therefore, steps 5 and 6 of the installation procedure for the `turnkey` version do not apply to the `developer` version. Detailed installation instructions are given in Appendix C. Appendix E has a list of resources about web development that we have found helpful.

Example

To demonstrate the use of the core version of the package, we present an example of a website that creates a plot of two variables from data in an uploaded .csv file. The website is pictured in Figure 6.

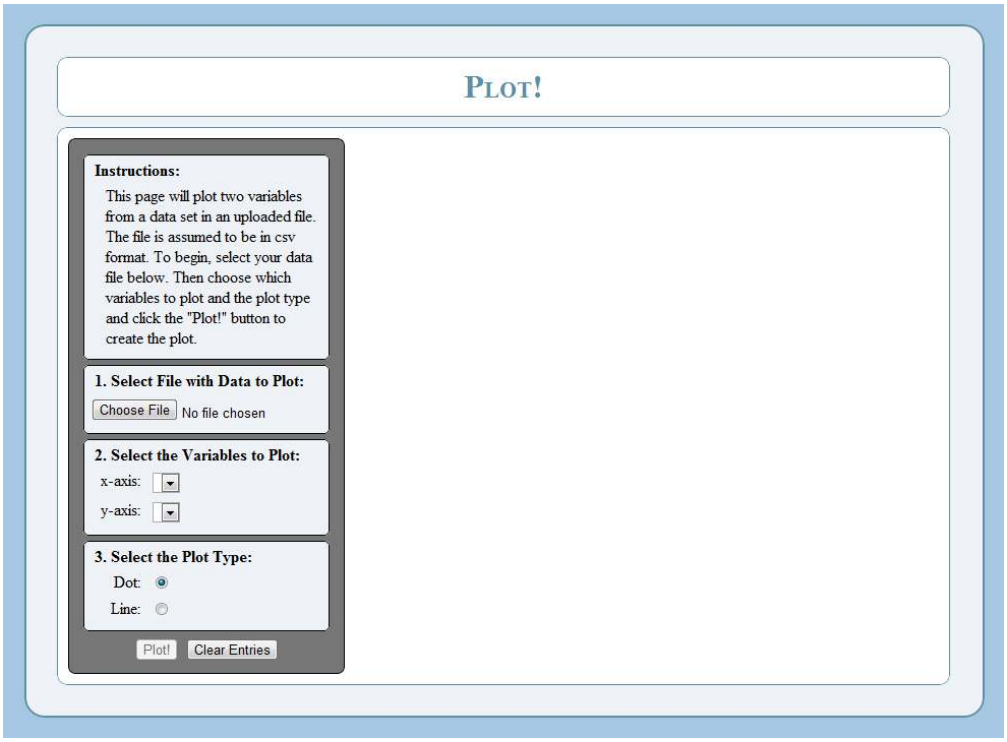

Figure 6: Screenshot of the example page for the `developer` version when the page is first loaded.

This website includes a couple of features that give a sense of the possibilities for smooth user interactions that can be created with AJAX. Specifically, user input can be processed as it is entered without bringing the user to a new page each time data is submitted to the web server. The first time we see this in action is when we populate the drop-down menus with options for the user to select which variables will be used in the plot. These options are automatically filled in when the user selects a data file, based on the variable names in the heading row of the file. In order to accomplish this, as soon as the user selects a data file, text that says "Loading..." is displayed and the file is uploaded to the server. On the server, the file is read in by an R script, and the variable names in the file are extracted and sent back to the user's web brower where they are inserted into the dropdown menus as shown in

Figure 7. Similarly, once the user selects the variables to plot and the plot type and clicks the "Plot!" button, these options are sent to a second R script which creates the plot. The plot is inserted into the same web page (Figure 8). A single user interface and a single web page is therefore maintained throughout the user's interaction with the web application.

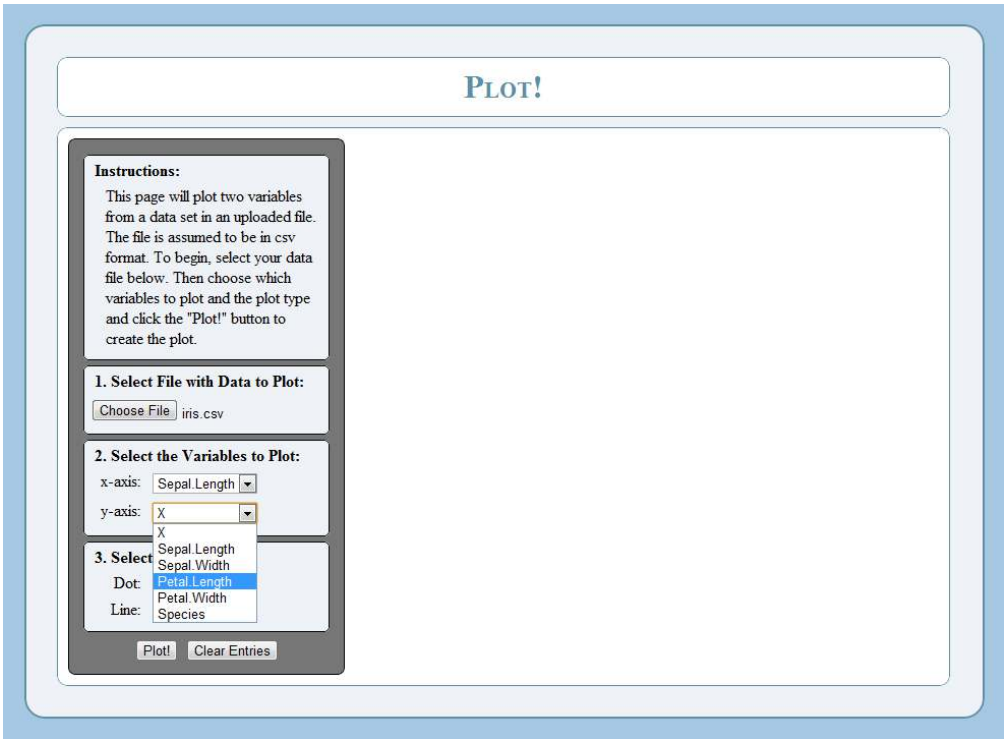

Figure 7: Screenshot of the example page for the `developer` version when the variable names have been loaded from the selected data file.

This website consists of 6 separate files:

- An HTML file with the content of the web page
- A CSS file to control the visual display of the web page
- A JavaScript file to handle user interactions with the page and form submissions
- The `WebDevelopR-dev.cgi` script included with **WebDevelopR**
- Two R scripts: one reads in the data file, sends the variable names back to the browser, and stores the data file on the server for later use by the second R script, and the second creates the plot.

These files are included in the installation directory of the package and are all thoroughly commented. The R scripts are also included in Appendix B of this paper for the reader's convenience.

The website has two different forms which are submitted to the server separately. The first form has only one user input field – the file selection field, which is named "`data`". This file

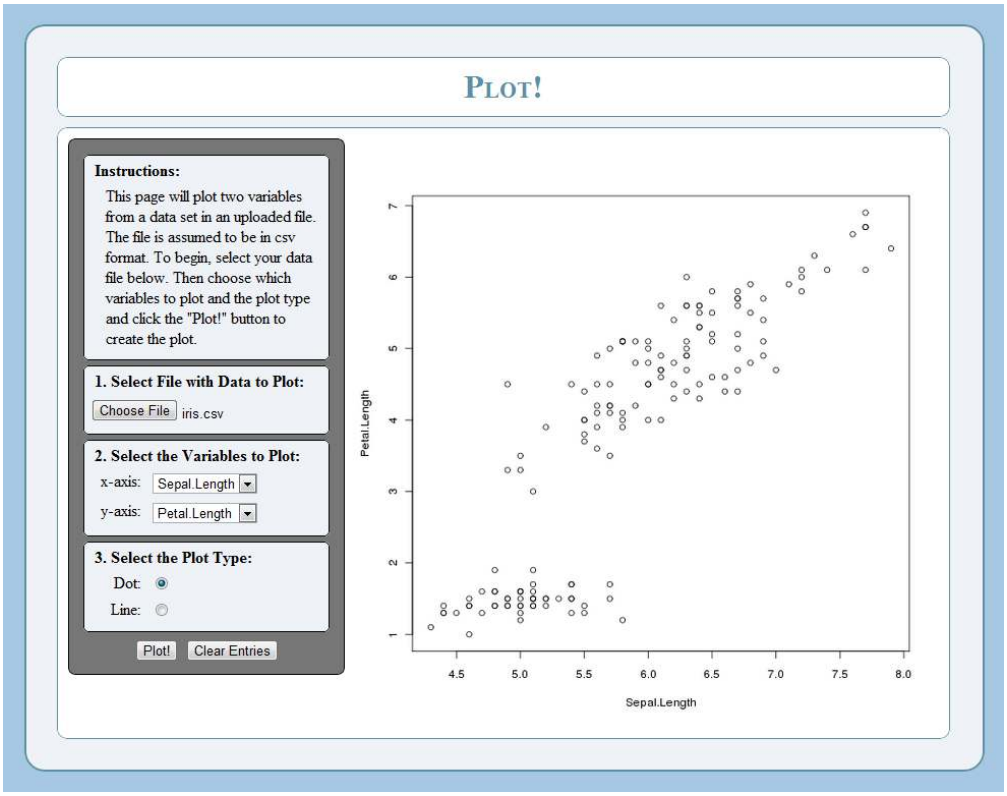

Figure 8: Screenshot of the example page for the `developer` version with the final plot.

selection field is set up so that whenever the user chooses a new file, the form is submitted to the server (i.e., the file the user selected is uploaded to the server).

Upon submission, the R script `example-dev1.R` is called to process the file upload. This script reads the uploaded data file and prints the variable names in the header row, as well as the session ID, in XML format. It also saves the data file on the server so that it can be used to create the plot later. The output from this script looks like the following:

```
<var1>Sepal.Length</var1>
<var2>Sepal.Width</var2>
<var3>Petal.Length</var3>
<var4>Petal.Width</var4>
<sessionID>163</sessionID>
```

When the R script finishes running, this output is sent back to the browser. A JavaScript function in the browser is then called to process the results. This function inserts the variable names into the drop-down menus in the second form, and creates a hidden field in the second form with the session ID from the first form submission.

The user can then select the variables to use in the plot and the plot type, and click the "Plot!" button. When this is done, the second of the two forms on the web page is submitted to the server and a different R script is called to process the submission. This script creates the plot specified by the user's selections and saves it as an image file on the server. It then

prints the HTML to include the image on the web page. The ouput from this script looks like the following:

```
<imgTagContainer>
<img src=\"/results/session163-image.png" width="100%" ></img>
</imgTagContainer>
```

When the R script exits, this output is sent back to the web browser. A JavaScript function is then called which inserts the image tag into the web page so that the plot is displayed to the user.

4. Some notes about security

There are several aspects of security to keep in mind when building web applications with **WebDevelopR** (or any other CGI scripting platform). These range from the "standard" concerns about preventing unauthorized access to your web server, to ensuring that some users do not use up so many computational resources that the server is unable to serve its intended clients, to securing the potentially confidential data that users upload so that anyone on the internet cannot access it. In this section we present an incomplete list of suggestions and thoughts on these topics. Several books on the subject are available (e.g., Sullivan and Liu (2011) and Hope and Walther (2008)), in addition to online resources such as the Open Web Application Security Project (`https://owasp.org`).

- Do not allow the web site user to enter R commands which will be executed on the server. This is dangerous if proper precautions are not taken because commands like `system` and `file.remove` can be used to alter important files on the server.

- Do not use input from the user to specify file names to read or write, or as a part of system commands. Again, this could allow a malicious user to modify important system files.

- As detailed in Appendix C, there are options in several of the CGI scripts to specify limits on the size of data uploaded through the form. It is recommended that you use these options to prevent disk space on the server from being used up.

- As mentioned above, Session IDs are generated sequentially, which means that they can be easily guessed. Additionally, transmissions between the server and the user's web browser are not encrypted and no checks are performed to ensure that the user accessing a particular file on the server (such as a graph or .csv file with analysis results) is the same one who submitted the data. This could be a problem if your users are submitting confidential or private data sets and do not want the results of the data analysis to be visible to others. To address this, you should change the method used to generate Session IDs and implement a secure connection between the web browser and the user's browser. Both of these changes are possible, but are beyond the scope of **WebDevelopR**.

5. Conclusions

It is common for statisticians to want to make analysis techniques they have developed available to others, including non-statisticians and people without programming expertise. Websites are a convenient way to achieve this goal. As a result, many solutions have been developed to facilitate the creation of websites that run R code. Each of these tools provides slightly different functionality, and makes a different trade-off between the ease of initial site creation and the ability to customize the site to suit project needs.

WebDevelopR encompasses two different solutions to this problem. One, the `developer` version, provides only the basic script required to facilitate communication between a site user's web browser and an R script. In many ways, this script can be viewed as an extension of **CGIwithR**, fixing a few technical problems associated with large data files and long computations, addressing some security issues, and allowing the web programmer to choose the data interchange format to be used in communications between the server and the browser. This version of the package puts the burden of creating the website with HTML, CSS, and JavaScript on the developer. However, it gives the developer the flexibility to create a variety of interactive user interfaces for the web application.

The `turnkey` version of the package builds on the basic functionality in the `developer` version, and includes a fully functional web site. This version allows statisticians with limited web development knowledge to create a web application that runs R scripts more easily than they could with other tools such as **rApache**, but with greater potential for customization than with packages such as **Rwui**.

Acknowledgments

This was partially supported by U.S. National Institutes of Health grants 1RC1HL099557 and RO1CA121005.

References

Banfield J (1999). "**Rweb**: Web-based Statistical Analysis." *Journal of Statistical Software*, **4**(1), 1–15. ISSN 1548-7660. URL http://www.jstatsoft.org/v04/i01.

Crane D, Pascarello E, James D (2006). *Ajax in Action*. Manning Publications.

Firth D (2003). "**CGIwithR**: Facilities for Processing Web Forms Using R." *Journal of Statistical Software*, **8**(10), 1–8. ISSN 1548-7660. URL http://www.jstatsoft.org/v08/i10.

Flanagan D (2011). *JavaScript: The Definitive Guide*. O'Reilly Media.

Hope P, Walther B (2008). *Web Security Testing Cookbook: Systematic Techniques to Find Problems Fast*. O'Reilly Media.

Horner J (2011). *brew: Templating Framework for Report Generation*. R package version 1.0-6, URL http://CRAN.R-project.org/package=brew.

Horner J (2013). ***rApache***: *Web Application Development with R and Apache*. URL http://www.rapache.net/.

Lecoutre E (2003). "The R2HTML Package." *R News*, **3**(3), 33–36. URL http://cran.r-project.org/doc/Rnews/Rnews_2003-3.pdf.

Meyer EA (2007). *CSS Web Site Design*. Peachpit Press.

Newton R, Wernisch L (2007). "**Rwui**: A Web Application to Create User Friendly Web Interfaces for R Scripts." *R News*, **7**(2), 32–35. URL http://www.r-project.org/doc/Rnews/Rnews_2007-2.pdf.

R Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Robbins JN (2006). *Web Design in a Nutshell*. O'Reilly Media.

RStudio, Inc (2013). ***shiny***: *Web Application Framework for R*. R package version 0.4.0, URL http://CRAN.R-project.org/package=shiny.

Sullivan B, Liu V (2011). *Web Application Security: A Beginner's Guide*. McGraw-Hill.

Urbanek S (2011). ***FastRWeb***: *Fast Interactive Framework for Web Scripting Using R*. R package version 1.0-1, URL http://CRAN.R-project.org/package=FastRWeb.

Urbanek S (2012). ***Rserve***: *Binary R server*. R package version 0.6-8, URL http://CRAN.R-project.org/package=Rserve.

Appendix

A. R Code for Turnkey Example

This is the R code used to process the form submission in the example of the `turnkey` version of the package in Section 3.1.

```
library("WebDevelopR")

############ Accessing the user input ###############

# The form in sample.html contains input fields named
# "data", "title", "description", "methods", and "plot".

submitted.data <- form.data()

file <- submitted.data[["data"]]
title <- submitted.data[["title"]]
```

```
description <- submitted.data[["description"]]
methods <- submitted.data[["methods"]]
plot <- submitted.data[["plot"]]

############ Using the utility functions ###########

# Status updates are shown in a separate section than the
# results.  The append=FALSE argument means that any previously
# logged status updates are removed.
status.update("Processing your submission...", append=FALSE)

if(sum(methods == "sd" | methods == "var") == 0) {
  status.update("Error: You must select a type of spread.")
  quit("no")
}

if(length(file.details()) == 0) {
  data <- data.frame(x = sample(1:5,5,TRUE),
    y = sample(1:5,5,TRUE), z = sample(1:5,5,TRUE))
  status.update("No file provided. Using random data...",
    append = TRUE)
} else {
  data <- read.csv(file)
}

# You can print strings directly to the results section using
# the cat function.
cat("This is the", title, "analysis.", description,
  "R used the following dataset:")

# web.print displays a nicely formatted version of R's print.
# The argument is a list of objects to print.
web.print(list(data))

results <- list()

# The elements of "methods" are taken from the "value"
# attributes of the inputs that the user selected (specified in
# the HTML code).
for(m in methods) {
  if(m == "mean") {
    results$mean = apply(data, 2,
      function(x) {mean(as.numeric(x))})
  } else if(m == "median") {
    results$median = apply(data, 2,
      function(x) {median(as.numeric(x))})
  } else if(m == "sd") {
```

```
    results$spread = apply(data, 2, sd)
  } else if(m == "var") {
    results$spread = apply(data, 2, var)
  }
}

results <- data.frame(results)

# The web.png and web.jpeg functions embed graphics into the
# results section displayed to the user.
if(plot == "png") {
  ### The alt text is displayed if the image cannot be viewed.
  web.png("results.png", attributes.text = "alt=\"Results\"",
    width = 600, height = 300)
  barplot(results$spread, names.arg = names(data),
    main = "column spread")
  garbage <- dev.off()
} else {
  web.jpeg("results.jpeg", attributes.text = "alt=\"Results\"",
    width = 600, height = 300)
  barplot(results$spread, names.arg = names(data),
    main = "column spread")
  garbage <- dev.off()
}

# web.csv creates a link to a file that the user can download.
web.csv(results, "results.csv", before.link.text = "Click ",
  link.text = "here",
  after.link.text = " to download the rest of your analysis.",
  open.new.window = FALSE, enclose.in.p = FALSE)

status.update("Analysis complete.", append = TRUE)
```

B. R Code for Developer Example

The example website for the `developer` version of **WebDevelopR** includes two R scripts. We provide the code for each of these scripts in this appendix. The first script reads in the uploaded data file, extracts the variable names from the column headers, and prints the variable names in an XML format. It also saves the data file so that the next script can use it to create the plot.

```
#####################################################################
##  Enter configuration information

# The location where the uploaded file will be stored
```

```
FILE_WRITE_DIR <- '/var/www/system/'

##  END configuration information entry
#####################################################################

library("WebDevelopR")

# The form in sample-core.html contains the input field
# "data".  This R script can access that input through the
# form.data or file.details functions.
# It can also access the sessionID through form.data.
submitted.data <- form.data()

sessionID = submitted.data[["sessionID"]]

# Save the data file so that it can be accessed by the second R
# script in order to create the plot. We put the session ID in
# the file name so that the file won't be overwritten by
# another user's submission.
file <- submitted.data[["data"]]
data <- read.csv(file)
save(data, file = paste(FILE_WRITE_DIR, 'session', sessionID,
  '-loaded-data.RData', sep=''))

# Output some XML with the names of variables in the uploaded
# data file. This will be used to populate the drop-down menus
# on the web page.
variables <- colnames(data)

for(i in 1:length(variables)) {
        cat("<var",i,">",variables[i],"</var",i,">\n",sep='')
}

# Output some XML with the sessionID.  We need to pass this on
# to the second R script so that it can read in the correct
# data file.
cat("<sessionID>",sessionID,"</sessionID>",sep='')
```

The second script creates the plot based on the user input and saves it as a graphic file on the server. It then prints HTML code to include the image on the web page.

```
#############################################################################
##  Enter configuration information

# The location where the uploaded file was stored
FILE_READ_DIR <- '/var/www/system/'
```

```
# The location where the plot should be stored
RESULTS_DIR <- '/var/www/html/results/'

# The relative web path to the location where the plot is stored
WEB_PATH_RESULTS_DIR <- '/results/'

##  END configuration information entry
################################################################################

library("WebDevelopR")

# The form in sample.html contains input fields for
# "xVar", "yVar", "method", and "firstSessionID".
# This R file can access these inputs through the elements
# associated with each of those names of the list returned by form.data.
submitted.data <- form.data()

# Read in data from the previously uploaded file.
load(paste(FILE_READ_DIR, 'session', submitted.data[["firstSessionID"]],
  '-loaded-data.RData', sep=''))

# Obtain the variables for the x and y axes of the plot selected
# by the user.
x_var <- submitted.data[["xVar"]]
y_var <- submitted.data[["yVar"]]

# Obtain the plot type for the plot selected by the user
if(submitted.data[["method"]] == "line") {
        method = 'l'
} else {
        method = 'p'
}

# Create the plot
png(paste(RESULTS_DIR, "session", submitted.data[["sessionID"]], "-",
  "image.png", sep = ""), width = 600, height = 600);
plot(data[, x_var], data[, y_var], type = method, xlab = x_var, ylab = y_var)
garbage <- dev.off()

# Output an image tag which will be inserted into the web page.
cat("<imgTagContainer>\n");
cat("<img src=\"", WEB_PATH_RESULTS_DIR, "session", submitted.data[["sessionID"]],
  "-image.png\" width=\"100%\" ></img>\n", sep = "");
cat("</imgTagContainer>");
```

C. Set Up and Configuration

There are seven major tasks to do in order to start using **WebDevelopR**: (1) Install a Perl interpreter and necessary Perl modules, (2) Install and configure a web server, (3) Install **WebDevelopR**, (4) Copy files included with **WebDevelopR** to appropriate locations on your server and set file permissions, (5) Edit configuration settings in CGI and Javascript files, (6) Modify the provided `index.html` file to include form elements for site users to enter input, and (7) Modify your R script to use submitted form data and provide output to site users. Each of these tasks is discussed in a Subsection of this Appendix. Note that the package has been tested with Linux and Windows; we expect that it would work with other operating systems as well, but have not tested this.

C.1. Install a Perl interpreter and necessary Perl modules

Perl interpreters are included with a standard installation of most Unix-like operating systems. If you do not have Perl, see your operating system documentation for instructions on how to install it. For Windows, you'll need to install an implementation of Perl such as Strawberry Perl (http://strawberryperl.com/) or ActivePerl (http://www.activestate.com/activeperl).

You will also need to ensure that the following Perl Modules are installed (typically, they will be by default):

- `CGI`
- `Fcntl`
- `HTML::Entities`
- `Time::HiRes`
- `threads`

You can check whether a Perl module is installed by entering `perl -MMODULENAME -e 1` at a command prompt (with "`MODULENAME`" replaced by the name of the module you are checking). For instance, to check whether the `HTML::Entities` module is installed, enter `perl -MHTML::Entities -e 1`.

If you don't see an error message in response to this command, the module is installed. If you do see an error message, you'll have to install the module. See the documentation at `http://www.cpan.org/misc/cpan-faq.html#How_install_Perl_modules` for instructions.

C.2. Install and configure a web server

There are many options for which web server you want to use, and installation and configuration details will vary with what solution you choose and which operating system you are using. The Apache HTTP server is one common choice; we provide brief instructions for installing and configuring it in a Windows environment here. Details for installation and configuration of Apache in a Unix environment depend on the version of Unix you are using; see documentation for your distribution for more guidance. Further documentation for Apache is also available online at `http://httpd.apache.org/docs/2.2/`, and more detailed instructions for installation under Windows are at `http://httpd.apache.org/docs/2.2/platform/windows.html`.

Please note that these instructions are intended to get you "up and running" quickly. We make no guarantees about the security of the configuration you'll end up with by following these instructions; you should read the documentation for more information.

Installation Instructions for Apache in a Windows Environment:

- First, download the software from `http://httpd.apache.org/download.cgi` In writing these instructions, I used the file "Win32 Binary without crypto (no mod_ssl) (MSI Installer): `httpd-2.2.19-win32-x86-no_ssl.msi`"; you could look for something similar. When you've downloaded the file, verify its integrity by following the instructions on the download page.
- Run the installation wizard and follow the prompts to complete the installation. In this process, consider choosing a different location to install the program to, such as `C:\\Apache`. This can prevent some problems with restrictive permissions that Windows sets for files and folders under `C:\\Program Files`.
- Edit configuration settings to enable CGI scripts.
  - Open the configuration file. In the Start Menu, find the "Apache HTTP Server 2.2" folder (the name may be slightly different), the "Configure Apache Server" subfolder, and choose "Edit the Apache httpd.conf Configuration File". You can edit this configuration file in a text editor such as notepad.
  - Make sure the line "`LoadModule cgi_module modules/mod_cgi.so`" appears in this file and is not commented out (i.e., there is not a '#' at the beginning of this line).
  - Make sure a line like "`ScriptAlias /cgi-bin/ "C:/Apache/cgi-bin/"`" appears in the file and is not commented out (i.e., there is not a '#' at the beginning of this line).
  - Note the locations of the `DocumentRoot` and the `cgi-bin` directory. You will copy the necessary files to these locations when you install **WebDevelopR** (see Section C.4 below).
  - Note the port that the web server is listening on, specified by the `Listen` directive. It is most convenient if this is set to 80.
  - Make sure the web server is started by clicking the "Start Apache in Console" shortcut in the Apache folder in the Start Menu. Then test it to make sure it is working by opening a browser and going to `http://localhost/`. You should see a default page provided by Apache. If you don't, see the documentation at `http://httpd.apache.org/docs/2.2/platform/windows.html` for a more detailed installation guide to troubleshoot.
  - Also test to make sure that your web server can run CGI scripts by going to `http://localhost/cgi-bin/printenv.pl`. (You may have to edit the first line of this script to point to the location of the Perl interpreter on your computer.) This is a simple CGI script provided by the Apache server which prints the environment variables. If you do not see any output from this script, see the documentation at `http://httpd.apache.org/docs/2.2/howto/cgi.html` for troubleshooting information.

C.3. Install WebDevelopR

Installing the package can be done as usual, with one caveat: if your webserver will run with a different username than yours (probably a good idea), you may need to install the package to a site library to ensure that an instance of R created by your web server will have access to the package. This can be achieved with the `lib` argument to `install.packages`. See `?libPaths` and `?install.packages` in R for more information.

C.4. Copy files and set file permissions

After you have installed **WebDevelopR**, you will need to copy several files from the installation directory of the package to locations where the web server will access them. The files to copy will be in either the `turnkey` or `developer` subdirectory of the package installtion directory, depending on which version of the package you are using. Each of these directories contains three subdirectories with files to copy: `cgi-bin`, `system`, and `html`.

`cgi-bin` **folder:** Copy the contents of the `cgi-bin` folder in the installation directory to your web server's `cgi-bin` folder. The files to copy depend on whether you are using the `turnkey` version or the `developer` version of **WebDevelopR**.

For the `turnkey` version:

- `displayResults.cgi`
- `preProcessForm.cgi`
- `processForm.cgi`
- `R.cgi`
- `statusCheck.cgi`
- Also place R script(s) you want the website to run in the `cgi-bin` folder. To try out our example, copy the file `example-turnkey.R`

For the `developer` version:

- `WebDevelopR-dev.cgi`
- Also place R script(s) you want the website to run in the `cgi-bin` folder. To try out our example, copy the files `example-dev1.R` and `example-dev2.R`.

You can typically find the location of the `cgi-bin` folder in your web server's configuration file. For instance, in Apache, this folder is specified by the `ScriptAlias` directive in the configuration file.

Set the permissions so that your web server can read and execute the files in the `cgi-bin` directory. On a Unix-like system, if your web server runs under a different username than yours the appropriate permissions are 755 for these files. For example, you can set the appropriate permissions for the `R.cgi` file by entering `chmod 755 R.cgi` at a command prompt when you are in the `cgi-bin` folder of your web server. In a Windows environment, these permissions

can be set in the dialog box brought up by right-clicking on the files and choosing "Properties". The specific permissions you need to choose depend on the version of Windows you are using, but the idea is to allow the web server to have read and execute permissions for these files while restricting other permissions as much as possible. On my version of Windows 7, that means setting *Write* permissions to "Deny".

On Unix-like systems, the permissions for the `cgi-bin` folder itself should be 711, so that the web server can access files in it (but it does not need to be able to list files there or write to the folder). On Windows 7, again we set *Write* permissions to "Deny". Once you have made these changes to the permissions, test running a CGI script such as `printenv.pl` (which is included with Apache) to be sure that the web server can run CGI scripts.

`system` **folder:** Create a directory **outside** of your web server's document root and the `cgi-bin` where "system" files can be stored and copy the contents of the system folder in the installation directory to it. Your web server should have read and write access to the files in this directory and read, write, and execute permissions for the directory; on Unix-like systems, this corresponds to permissions of 666 for the files and 777 for the directory.

`html` **folder:** If you are using the `developer` version of the package, you will be writing your own HTML files, so you don't need to copy any files here unless you want to view our examples. If you want to try out the examples for the `developer` version of the package or if you are using the `turnkey` version, copy the contents of the `html` folder in the appropriate subdirectory of the installation directory of **WebDevelopR** to your web server's `html` folder (or equivalent – for the Apache server on Windows, this folder is named `htdocs`). You can typically find the location of the `html` folder in your web server's configuration file. In Apache, this folder is specified by the `DocumentRoot` directive in the configuration file.

Set the permissions of the `.html` files in the `html` folder, the `.jpg` and `.gif` files in the `images` subfolder, the `.js` files in the `scripts` subfolder, and the `style.css` file in the `style` subfolder so that the web server has read access to them. On Unix-like operating systems, the appropriate permissions for these files are given by 644. On Windows 7, set *Write* permissions for these files to "Deny". On Unix, the web server needs to have execute permissions for these folders so that it can access files in them; the permissions for these folders should be set to 711. On Windows, deny write access to these folders.

If you are using the complete version of the package and your R script will be writing files for the site user to view, such as images or .csv files, the web server should have write and execute permissions for the `results` subfolder of the `html` folder. On Unix, the appropriate permissions are given by 733, and on Windows 7 no changes should be made to the permissions. If your script does not need to create output files, you can remove this folder.

C.5. Edit Configuration Settings

Configuration settings are located at the beginning of the `.cgi` files in the `cgi-bin` folder, and (for the `turnkey` version of the package) in the JavaScript files `formSubmitScript.js` and `resultsScript.js` in the `html/scripts` folder, as well as the `index.html` file in the `html` folder. A list of all files in which configuration settings should be edited and the meaning of those configuration settings is below. Note that many of these configuration settings have

default values that will work for most users, and therefore may not need to be edited. These settings are indicated with a "*(Default value OK)*" flag.

For the `developer` version, all configuration settings are in the file `WebDevelopR-dev.cgi` in the `cgi-bin` directory. The configuration settings in this file are:

- On the first line of the script, make sure the location of the Perl interpreter is specified correctly. The format should be something like `#!  /usr/bin/perl -wT` on Unix-like operating systems, or `#!  c:/strawberryPerl/perl/bin/perl.exe -wT` on Windows.
- `$CGI::DISABLE_UPLOADS` This setting determines whether file uploads are disabled (1) or enabled (0). It is recommended that you disable uploads if they are not required for your web application. This can prevent denial of service attacks on your server in which many files are uploaded.
- `$CGI::POST_MAX` The size restriction (in bytes) for the total upload including any files uploaded and other data entered or selections made in the web form.
- `$system_path` The full local server path to the directory where "system" files are stored. The web server should have read/write access to this directory, but it should not be viewable over the internet (i.e., it should be located outside the document root of your web server).
- `$R_path` The full local server path to R, including the executable. On a Linux machine, this might be something like `/usr/bin/R`
- `$GS_path` The full local server path to Ghostscript, including the executable. On a Linux machine, this might be something like `/usr/bin/gs`
- `$R_scripts` The name of the R script(s) you want to run. If you want to run more than one script, enter them in a string separated by commas (but no whitespace).
- `$data_type` The data type to be returned to the user's browser. Should be `"HTML"`, `"XML"`, or `"JSON"`.
- `$new_path` The value the `PATH` environment variable will be set to before R is called (this is done for security reasons). On Linux, this can be an empty string; on Windows, it should include `C:\\Windows\\system32`.

For the `turnkey` version of the site, the same configuration settings for the CGI scripts often appear in multiple files. The following table lists each configuration setting for the CGI scripts, the files they appear in, and gives a description of the setting. In addition to these settings, make sure the location of the Perl interpreter is specified correctly on the first line of each script. The format should be something like `#!c:/strawberryPerl/perl/bin/perl.exe -wT` on Windows, or `#!  /usr/bin/perl -wT` on Unix-like operating systems.

Table 3: Configuration settings in CGI scripts for the complete version of **WebDevelopR**

Variable Name	Files to Edit	Description of Setting
`$CGI::DISABLE_UPLOADS`	`preProcessForm.cgi` `processForm.cgi`	This setting determines whether file uploads are disabled (1) or enabled (0). It is recommended that you disable uploads if they are not required for your web application. This can prevent denial of service attacks on your server in which many files are uploaded.
`$CGI::POST_MAX`	`preProcessForm.cgi` `processForm.cgi`	The size restriction (in bytes) for the total upload including any files uploaded and other data entered or selections made in the web form.
`$GS_path`	`processForm.cgi` `R.cgi`	The full local server path to Ghostscript, including the executable. On a Linux machine, this might be something like `/usr/bin/gs`
`$new_path`	`processForm.cgi` `R.cgi`	The value the `PATH` environment variable will be set to before R is called (this is done for security reasons). On Linux, this can be an empty string; on Windows, it should include `C:\\Windows\\system32`.
`$page_header_text`	`displayResults.cgi` `processform.cgi`	Header text for the web page – appears in large lettering at the top of the web page.
`$R_path`	`processForm.cgi` `R.cgi`	The full local server path to R, including the executable. On a Linux machine, this might be something like `/usr/bin/R`
`$R_script`	`processForm.cgi` `R.cgi`	The name of the R script you want to run.
`$results_page_title`	`displayResults.cgi` `processform.cgi`	The title of the web page – typically displayed in the bar at the top of the user's web browser.
`$results_path`	`processForm.cgi` `R.cgi`	The full local server path to the `results` subfolder of the `html` directory.

Table 3: CGI script configuration settings (continued)

Variable Name	**Files to Edit**	**Description of Setting**

`$system_path`	`displayResults.cgi`
	`preProcessForm.cgi`
	`processForm.cgi`
	`R.cgi`
	`statusCheck.cgi`
		The full local server path to the directory where "system" files are stored, as created in Subsection C.4 above.

`$cgibin_web_path`	`displayResults.cgi`	*(Default value OK)* The relative web path to the cgi-bin directory, where CGI scripts for the website are stored – probably something like `/cgi-bin/`
`$images_web_path`	`displayResults.cgi` `processform.cgi`	*(Default value OK)* The relative web path to the images directory, where image files for the website are stored – probably something like `/images/`
`$results_web_path`	`processForm.cgi` `R.cgi`	*(Default value OK)* The relative web path to the results directory, where any files you create for the website user during your analysis are stored (e.g. image and/or .csv files) – probably something like `/results/`
`$scripts_web_path`	`displayResults.cgi`	*(Default value OK)* The relative web path to the style directory, where .js javascript files for the website are stored – probably something like `/scripts/`
`$style_web_path`	`displayResults.cgi` `processform.cgi`	*(Default value OK)* The relative web path to the style directory, where .css style files for the website are stored.

There are also some configuration settings in the JavaScript files in the `html/scripts` directory:

- formSubmitScript.js
  - `cgibinDir` *(Default value OK)* The relative web path to the cgi-bin directory, where CGI scripts for the website are stored – probably something like `/cgi-bin/`
- resultsScript.js
  - `statusCheckFrequency` *(Default value OK)* The number of seconds to wait between polling the server for status updates.

– `imagesDir` *(Default value OK)* The relative web path to the images directory, where image files for the website are stored – probably something like `/images/`

C.6. Edit/create HTML file(s)

The next step is to create the HTML pages for your site users to interact with your application. For the `turnkey` version of the package, you can do this by modifying `index.html` to provide the necessary form elements and instructions. This file has examples of all standard HTML form elements for your reference. Setting up your document with a similar structure to the example will allow predefined style rules to lay out the form consistently. Logically related form elements should be contained within a `<fieldset>...</fieldset>` tag, instruction text should be enclosed in a `<p>...</p>` tag, and each form element should consist of a `<label>...</label>` tag and some sort of form input element, both enclosed within a `<div class="formItem">...</div>` tag. For more thorough documentation and examples of form input elements, see `http://www.w3schools.com/html/html_forms.asp`.

There are also several other aspects of the included `index.html` file to modify:

- On line 15, edit the title text. This text is typically displayed in the bar at the top of the user's web browser.
- *(Default value OK)* On line 21, if necessary, edit the relative web path for the `style.css` file.
- *(Default value OK)* On lines 27 - 29, if necessary, edit the relative web paths for the javascript files.
- On line 40, edit the header text. This text is displayed in large text near the top of the web page.
- *(Default value OK)* On line 58, if necessary, edit the relative web path for the processForm.cgi script.

C.7. Modify your R script

You will need to update your R script in two ways.

First, you will want to make use of any data or options specified by the user in the web form. You can access this information via the `form.data` and `file.details` functions, as discussed in Section 3.1.1 above. Note that no checks have been done to ensure that form data exists or is what you were expecting (e.g., that the user didn't actually type letters into a field where you expected them to enter a number). You should explicitly check this in your R script and provide feedback to the user if something was entered incorrectly.

Second, you will typically want to display results of the analysis to the web site user. As discussed above, the package provides some utility functions to make it easier to embed images and links to data files in the web page, as well as to directly print R objects. In addition to this, you can use the `cat` function to print text (including HTML tags) to the user's browser. To ensure that all browsers process your output correctly, be sure that your HTML output is well-formed (i.e. that each opening tag has a matching closing tag and that

a tag that is opened inside of another tag is closed before the outer tag). See the example script in Section 3.1.3 for how this works.

Your R script should be placed in the `cgi-bin` directory of your web server.

D. Utility Functions

We have provided a few utility functions that you can use in your R scripts to perform common tasks. In this appendix we document these functions. See also Section 3.1.3 for an example with some context. Note that these functions are only relevant to the "complete" version of the package, since they rely on the structure of this particular web application. However, the code may be useful as a reference to developers who encounter similar needs when using the "core" version.

- `status.update`

  Description

  Used to provide status updates to the user.

  Usage

  ```
  status.update(status, append=FALSE)
  ```

  Arguments

  `status` The status message to be displayed to the site user.

  `append` Should the status be added to previous status updates, or should it replace previous status updates?

  Details

  This function writes status update text to the session-specific temporary file used for storing status updates. These updates are automatically retrieved and displayed by the user's browser while the R script is running.

  Value

  None

  Examples

  ```
  status.update("Processing your submission - Stage 1")

  status.update("Processing your submission - Stage 2", append=TRUE)
  ```

- `web.jpeg` and `web.png`

  Description Used to create .png and .jpeg image files (respectively), and to embed these images in the results page.

  Usage

  ```
  web.jpeg(file.name="image.jpeg", attributes.text="", centered=TRUE, ...)
  ```

  ```
  web.png(file.name="image.png", attributes.text="", centered=TRUE, ...)
  ```

  Arguments

`file.name` The name of the image file. Note that `file.name` will be pre-pended with "session", the session ID, and a dash. For example, if you call `web.jpeg` with the default argument of `file.name = "image.jpeg"`, and the session ID is 23, the file name actually used for the image will be "`session23-image.jpeg`".

`attributes.text` This text will be used for the attributes text of the HTML image tag.

`centered` Indicates whether the image should be centered when it is displayed to the user in the results page.

`...` Arguments to `jpeg` or `png`, respectively.

Details

These functions are wrappers for `jpeg` and `png`. The functions store the image file in the results directory on your web server (this directory is specified in the configuration section at the top of the `processForm.cgi` and `R.cgi` CGI scripts). They also create the necessary HTML tags to insert the image into the results page that is displayed to the site user.

Note that in order to allow the web site user to access these files, they will not be deleted automatically. You should periodically review files in the results directory and clear old ones out.

Value

None

Examples

```
web.jpeg("plot.jpeg", attributes.text="alt=\"Your Plot\"", width=600,
    height=300)
plot(x=rnorm(10), y=rnorm(10))
points(x=0, y=0, pch="+", col="red")
garbage <- dev.off() # capture the output from calling dev.off to keep
                     # it from being displayed in the results page

web.png("plot.png", attributes.text="alt=\"Your Plot\"", width=600,
    height=300)
plot(x=rnorm(10), y=rnorm(10))
points(x=0, y=0, pch="+", col="red")
garbage <- dev.off() # capture the output from calling dev.off to keep
                     # it from being displayed in the results page
```

- `web.print`

Description

Prints objects embedded in a nicely formatted `<pre> ... </pre>` HTML environment (similar to verbatim in LaTeX).

Usage

```
web.print(objects, width = 80, leading.spaces.num = 2,
    continued.line.indent.num = 6)
```

Arguments

`objects` A list of objects to print. (Note that objects is iterated over and every component is printed; if you don't embed the objects you want to print in a list, this could result in some funny behavior. This is done to allow you to print multiple objects within the same shaded area on the results page.)

`width` Number of characters per line (not including `leading.spaces.num` but including `continued.line.indent.num`).

`leading.spaces.num` Number of spaces to insert before each line. These spaces create some padding in the user display.

`continued.line.indent.num` If a line of more than `width` characters is broken, the number of extra spaces to insert before the continued line.

Details

`web.print` allows you to send the result of a call to the `print` function to the user's web browser. The output is embedded in a `<pre>` tag, which is displayed in the browser with a fixed width font and a shaded background.

Value

None

Examples

```
data(iris)
web.print(iris$Petal.Length) #each vector element is on a separate line
web.print(list(iris$Petal.Length)) #probably want you wanted

fit <- lm(Petal.Length ~ Petal.Width + Sepal.Length, data=iris)
web.print(list(summary(fit), summary(fit)$cov.unscaled))
```

- `web.table, web.csv, and web.csv2`

Description

Prints a data frame or matrix to a .csv format file and inserts a link to the file in the results page displayed to the website user.

Usage

```
web.table(x, file.name = "table.txt", before.link.text="Click ",
    link.text="here",
    after.link.text=" to download your file.  (The link opens in a new
    window or tab; alternatively, you can right-click or option-click on
    the link and choose "Save As..." to download the file.)",
    open.new.window=TRUE, attributes.text="", enclose.in.p = TRUE, ...)

web.csv(x, file.name = "table.csv", before.link.text="Click ",
    link.text="here",
    after.link.text=" to download your file.  (The link opens in a new
    window or tab; alternatively, you can right-click or option-click on
```

```
    the link and choose "Save As..." to download the file.)",
    open.new.window=TRUE, attributes.text="", enclose.in.p = TRUE, ...)

web.csv2(x, file.name = "table.csv", before.link.text="Click ",
    link.text="here",
    after.link.text=" to download your file.  (The link opens in a new
    window or tab; alternatively, you can right-click or option-click on
    the link and choose "Save As..." to download the file.)",
    open.new.window=TRUE, attributes.text="", enclose.in.p = TRUE, ...)
```

Arguments

`x` The object to be written to a file. This is passed on to the write.table, write.csv, or write.csv2 function, which prefer that `x` be a matrix or data frame. If it is another data type, an attempt is made to coerce `x` to a data frame.

`file.name` The name of the output file. Note that `file.name` will be pre-pended with "session", the session ID, and a dash. For example, if you call `web.csv` with the default argument of `file.name = "table.csv"`, and the session ID is 23, the file name actually used for the image will be "`session23-table.csv`".

`before.link.text` Text to be displayed in the results page immediately before the link.

`link.text` The clickable text displayed in the results page as a link to the output file.

`after.link.text` Text to be displayed in the results page immediately after the link.

`open.new.window` Indicates whether the link should be set to open in a new window. The default is `TRUE`; this prevents users from accidentally opening the csv file in the same screen and then having to re-submit their data for analysis to see other results. Note that this option works by appending `target="_blank"` to the attributes text.

`attributes.text` Attributes text for the HTML link.

`enclose.in.p` If `TRUE`, `before.link.text`, the link, and `after.link.text` are all placed within a paragraph HTML element. Otherwise, this text is not placed within a block level HTML element.

`...` Arguments to `write.table`, `write.csv`, or `write.csv2`, respectively.

Details

These functions are wrappers for `write.table`, `write.csv`, and `write.csv2`, respectively. The functions store the output file in the results directory on your web server (this directory is specified in the configuration section at the top of the `processForm.cgi` and `R.cgi` CGI scripts). They also create the necessary HTML tags to insert a link to the output file into the results page that is displayed to the site user.

Note that these files will not be deleted automatically. You should periodically review files in the results directory and clear old ones out.

Value

None

Examples

```
data(iris)
web.table(iris, "iris.txt")
web.csv(iris, "iris.csv")
web.csv2(iris, "iris2.csv")
```

E. Resources for Web Development

In this Appendix we list a few resources that we have found helpful in learning about web development, although whatever is available at your local book store will probably suffice too.

The following websites cover web technologies like HTML, CSS, and JavaScript:

- `http://www.webplatform.org/`
- `http://reference.sitepoint.com/css`

There are also many JavaScript libraries that make developing web applications much easier. One popular alternative is the jQuery library. Tutorials and documentation for jQuery are online at `http://jquery.com/`. An extensive list of alternatives to jQuery is available on Wikipedia at `http://en.wikipedia.org/wiki/List_of_JavaScript_libraries`.

We have found the books Web Design in a Nutshell (Robbins 2006) and CSS Web Site Design (Meyer 2007) to be helpful in learning about HTML and CSS. JavaScript: The Definitive Guide (Flanagan 2011) discusses JavaScript at an intermediate to advanced level. Ajax in Action (Crane, Pascarello, and James 2006) discusses all aspects of AJAX, including what AJAX is, what can be accomplished with it, how web applications using AJAX can be organized, an introduction to JavaScript, and a discussion of server-side technologies.

The R packages **R2HTML** (Lecoutre 2003) and **brew** (Horner 2011) can be helpful in generating HTML from R.

Affiliation:

Evan L. Ray
Department of Mathematics and Statistics
University of Massachusetts, Amherst
Lederle Graduate Research Tower, 710 North Pleasant Street
Amherst, Massachusetts 01003, U.S.A.
E-mail: ray@math.umass.edu