The **SRPM** Package

Roger D. Peng
Department of Biostatistics
Johns Hopkins Bloomberg School of Public Health

May 30, 2007

1 Introduction

The **SRPM** package ("Shared Reproducibility Package Management") provides tools for creating and interacting with what we call "shared reproducibility packages". These shared reproducibility packages (SRPs) are *not* true `R` packages but rather contain information related to a Sweave document that can be distributed to a wide audience and used to reproduce the results. The format of an SRP is meant to be simple so that it can be used on different systems.

Each package is simply a directory which contains the following subdirectories:

- `article/`: contains the original Sweave file and a "weaved" version of the file (e.g. in PDF format)
- `figures/`: contains files corresponding to any figures in the document
- `cacheDB/`: contains a collection of **stashR** databases storing any cached computations from code chunks
- `code/`: contains code files corresponding to the `R` code for each code chunk in the Sweave document (these are produced with the `Stangle` function with the argument `split = TRUE`).

Each SRP also has a file called `metadata.dcf` which is a text file containing information about each of the code chunks in the document. This file is written in the Debian Control File format.

In addition, a package may contain a file named `REMOTE` which contains the URL of the location of any remote **stashR** databases containing cached computations. If the size of the cached computation databases is large, an author may wish to post them on a webserver rather than distribute them with the SRP itself. The `REMOTE` file indicates the location of the cached computation databases and configures the other tools in the **SRPM** package to retrieve data from this location using the functionality in the **stashR** package.

2 Author tools

A shared reproducibility package can be constructed with the `makeSRP` function from the **SRPM** package which takes as arguments the name of the package to create and the name of the Sweave file for the original document. The `makeSRP` function

1. creates the necessary directories for the SRP;

2. calls `Stangle` to create individual code files for each code chunk and copies the files into the `code/` subdirectory;

3. copies the **stashR** databases containing the cached computations into the `cacheDB/` subdirectory;

4. copies graphics files corresponding to figures into the `figures/` subdirectory;

5. copies the article PDF file and Sweave file into the `article/` subdirectory;

6. creates the metadata file by reading the map file produced by the **cacheSweave** package and writes it to the `metadata.dcf` file.

Currently, the **SRPM** package requires that both the graphics files for the figures and the weaved version of the article be in PDF format, however we hope to remove this limitation in the near future.

Another function that is available to authors is the `makeWebpage` function which produces a simple webpage corresponding to an SRP. The webpage lists all of the code chunks in a document with links to the code itself. Also, there are links to cache databases as well as the PDF versions of figures so that readers can browse an SRP using a web browser and without having to have R installed.

3 Reader tools

Sweave is an example of a tool that is useful to authors of statistical or scientific documents in that it assists in the development of documents by ensuring that the text and data analysis are closely integrated into a single document. However, readers of reproducible documents also need tools to assist them with interacting with the data analyses therein and reproducing key results.

In the **SRPM** package we provide some basic tools for readers of Sweave documents that allow them to interact with the code and data provided by a shared reproducibility package created by the author. The basic functions are

- `code`: When called with no arguments, a listing of all the code chunks in the article is printed to the console. The `code` function can also take a numeric argument corresponding to the code chunk sequence number of a character argument corresponding to the code chunk name. When `code` is passed a numeric or character argument, it returns an object of class "codeObject" which contains the code and pointers to any cached computation databases or figures associated with the code chunk.

- `article`: This function takes no arguments; when called it launches the article PDF document in the PDF viewer.

- `figure`: This function must be given a numeric argument corresponding to the figure number in the original article. When called, it displays in the PDF viewer the figure corresponding to the figure number.

- `cache`: This function takes a code chunk sequence number or a code chunk name (character) as an argument and returns an object of class "localDB" or "remoteDB" depending on whether the SRP is using local or remote cached computation databases. This object can be explored with the methods defined in the **stashR** package (see also details in Eckel and Peng, 2006).

- `loadcache`: This function lazy-loads cached computation databases into the global environment. It takes a numeric vector of code chunk sequence numbers or a character vector of code chunk names and loads the cached computation databases associated with those code chunks in the order that they are specified. Once a database is lazy-loaded, the object names appear in the environment into which the database was loaded, but they do not occupy any extra memory until they are first accessed. If a specified code chunk does not have a database associated with it, no action is taken.

- `runcode`: The `runcode` function takes as input a numeric vector of code chunk sequence numbers or a character vector of code chunk names and executes the code in those code chunks. Each code chunk is evaluated in the order in which it appears in the input vector. By default, if a cached computation database is associated with a code chunk, then the database is lazy-loaded via `loadcache` rather than executed. In order to force evaluation of code in a code chunk with a cache database, one needs to set `useCache = FALSE` when calling `runcode`. If an error occurs when executing the code in a code chunk, a message is printed to the console indicating the error and the code chunk is skipped.

- `edit`: A method is provided for the `edit` generic function for objects of class "codeObject" which can be used to edit the R code corresponding to a code chunk. The modified "codeObject" object can be executed with the `runcode` function. The `edit` method returns the modified copy of the object so that the original code is not modified. The editor used is that which is launched by the `file.edit` function and will be system dependent.

These functions consist of the primary user interface for readers to interact with shared reproducibility package. Certain SRPs may also require that other R packages be installed in order to execute the code in the code chunks and these should be installed before attempting to execute the code with `runcode`.

Other utility functions available to the user are `currentPackage`, which shows the currently registered SRP, `getRemoteURL`, which returns the URL of the remote cached computation databases (if any), and `getLocalDir`, which returns the path to the directory where local copies of the remote cache databases will be stored.

4 Example

The **SRPM** package depends on the **methods** and **stashR** packages and additionally imports the **utils**, **filehash**, and **cacheSweave** packages. Once those dependencies are installed, the **SRPM** package can be loaded using `library` in the usual way.

The first thing a user must do is register a shared reproducibility package (SRP) using the `setPackage` function. We will use as an example a simple SRP called `srp_simple` that comes with the **SRPM** package. The package can be registered by passing the name of the directory to the `setPackage` function.

```
> library(SRPM)
> pkgdir <- system.file("SRP-ex", "srp_simple", package = "SRPM")
> setPackage(pkgdir)
```

Upon registering a package one can call the `article` function (with no arguments) to open a PDF copy of the full article in the PDF viewer (as identified by `getOption("pdfviewer")`). Another useful function to begin with is the `code` function. Called with no arguments, `code` lists all of the code chunks in the article.

```
> code()
```

```
1 LoadPackages
2 FitLinearModel  [C]
3 CoefficientTable
4 LinearModelDiagnosticPlots [Figure 1]
```

The code chunk listing is annotated with three different types of tags. The first is the code chunk sequence number which appears to the left of the code chunk name. This number can be used to identify a code chunk in other operations. The second is a `[C]` which indicates caching has been turned on and that the corresponding code chunk gives rise to a cached computation database. Lastly, code chunks with the tag `[Figure ?]` produce figures or plots. For example, code chunk 4 produces Figure 1 in the original article.

Code in any of the code chunks can be executed with the `runcode` function by passing the code chunk name or sequence number. Sequences of code chunks can be executed by passing a numeric or character vector to `runcode`. For example, to execute the code in chunks 1 through 3 to create the table of regression coefficients, we can execute

```
> runcode(1:3)
```

which prints the following messages

```
running code in code chunk 1
loading cache for code chunk 2
running code in code chunk 3
```

and prints a table to the console (here, using the **xtable** package). Code chunk 2 has a cached computation database associated with it so the database is lazy-loaded into the workspace instead of the code being executed. Code chunk 3 creates the table and it is executed successfully.

In order to explicitly lazy-load a cached computation database into the workspace one can use the `loadcache` function. For example, to load the database for code chunk 2, one can call

```
> loadcache(2)
> ls()
```

```
[1] "airquality" "fit"        "pkgdir"
```

The "localDB" object representing the cached computation database can be explicitly retrieved by calling the `cache` function and accessed using the methods defined in the **stashR** package. The above code fragment is roughly equivalent to

```
> library(stashR)
> db <- cache(2)
> show(db)
```

```
'localDB' database 'FitLinearModel'
```

```
> dbList(db)
```

```
[1] "airquality" "fit"
```

```
> dbLazyLoad(db)
```

5 Package websites

If the author has created a webpage via the `makeWebpage` function, the reader may prefer to view that first. The webpage created by `makeWebpage` corresponding to the `srp_simple` package can be found at

`http://www.biostat.jhsph.edu/~rpeng/RR/ea6883e28967ec48a0ee009585da0eb0/html/`

Currently, the webpage created by `makeWebpage` resembles the output provided by the R functions at the console. The cached computation databases can be browsed in a limited fashion—smaller objects can be viewed in their entirety while for larger objects a summary is provided by showing the output from `str`.

References

Eckel SP, Peng RD (2006). "Interacting with local and remote data repositories using the stashR package for R." *Technical Report 127*, Johns Hopkins University Department of Biostatistics. http://www.bepress.com/jhubiostat/paper127.