JSweave User Manual

Wataru Shito

Seinan Gakuin University

Fukuoka – Japan

Version 0.2

Documentation Revision: 1.8

Project Homepage: http://www.seinan-gu.ac.jp/~shito/jsweave

Maintainer: Wataru Shito (shito@seinan-gu.ac.jp)
Bug reports and feature requests are welcome!
Feel free to send any comments too!

1 Introduction

JSweave is a free Java program (Apache License) that produces a LaTeX document combined with the output from R, a statistical package. By embedding the R source code into a LaTeX document with a markup tag, JSweave will insert an R output and create a new LaTeX document. The graphs are inserted automatically as well. JSweave can also be used to extract the R source code from the document.

The combination of a documentation and programming code for readability is called literate programming named by Donald Knuth in his book. The statistical package R provides a literate programming tool for S language called Sweave. JSweave is just a superset of Sweave and provides additional features. A summary of the extended features are as follows.

1. Flexible code reuse.

   In Sweave, the smallest unit of a reused code chunk is limited to a statement since it is implemented with R. So, the partial code within the brace in a definition of a function cannot be an individual code chunk. However, this limit is eliminated in JSweave.

   Another extension in code reuse of JSweave is that in order to increase the readability as well as to save space, you can choose whether or not to expand the reused code. This is one of the useful features in literate programming.

2. Presenting the source style without the input prompt.

   Sometimes, we wish to execute R code and to only display the source code without the R input prompt. JSweave provides another verbatim style for this which is useful when writing a long definition of a function.

3. Compact verbatim format.

JSweave uses a more compact form of the verbatim format. Sweave puts extra gaps between inputs and outputs. In JSweave, it does not insert any vertical gaps between inputs and outputs as long as they belong to the same code chunk. JSweave also takes the typewriter style for all inputs and outputs in order to keep the same appearance as the R console. However, in the source type presentation without the R input prompt, the comments are displayed in slanted text the popular format found in S language text books.

In future releases, the choice of styles between JSweave and the original Sweave will be left to the user.

4. Internationalization support.

   JSweave accepts different character encodings. R supports UTF-8 since R 2.1.0[1]. JSweave converts the character encoding of the source document into UTF-8 and produces the LaTeX document in the encoding in which the source document was written. Hence, you can write documents with any encodings and obtain nice statistical reports in your language.

Because JSweave is an extention of Sweave, process the original Sweave document and see the difference in the output.

2 Requirements

To run JSweave, you need a shell (sh) and a Java run time environment later than version 1.5.0 as well as R later than version 2.1.0.

At the moment, JSweave only provides the final LaTeX document under a UNIX like environment (including MacOS X) since JSweave requires a shell program. I have not checked with cigwin under Windows.

To build the binary from the source you need Java SDK and Jakarta Ant. This official build is done under Java SDK 1.5.0 with Jakarta Ant 1.6.2. There should be no problem if the environment has newer versions of these.

3 Install

Place the jar file, `jsweave.jar` anywhere you want and invoke

```
$ java -jar {PATH_TO_JAR}/jsweave.jar input-file.Rnw
```

It will create `input-file.tex` in a current directory.

The ready-to-use jar file must be available with the distribution however, if you want to build one from scratch, simply type

```
$ ant jar
```

This will create `jsweave.jar`.

[1] See http://developer.r-project.org/Encodings_and_R.html.

4 Usage

First of all, you need to create a document to process with JSweave. The file name of the document has to have '.Rnw' suffix. The basic document structure is the same as Sweave where you insert the R code in LaTeX document marked up with certain tags. At the beginning of the R code, insert a line with `<< >>=`. At the end of the code insert a line with an @ mark. It should look like this:

```
<< >>=

# R code

@
```

This is all you need to see how JSweave works.

If you want, you can also label the code chunk or set some options inside the beginning tag as

```
<<Chunk name, optionA=optionValue, optionB=optionValue >>=
```

Please see the Sweave manual[2] for the basic options. In the following sections, this manual will only explain the extended options. For those already familiar with Sweave, it would be a good idea to process the Sweave document with JSweave to see the difference. Then, you can dig into some of the extra options available in JSweave[3].

Once the Rnw document is ready, you can run JSweave as follows.

```
$ java -jar {PATH_TO_JAR}/jsweave.jar input-file.Rnw
```

This will create `input-file.tex` and you can compile this with LaTeX.

If the character encoding of your input file is different from the system default one, use the '`-encoding`' option. For example, if your document is written in Japanese EUC encoding, you can type the following lines at the shell prompt.

```
$ export LC_ALL=ja_JP.UTF-8
$ java -jar {PATH_TO_JAR}/jsweave.jar -encoding EUC-JP input-file.Rnw
```

The first command sets the locale with UTF-8 since R only supports UTF-8 for multi-byte characters. The second command is for JSweave. This will convert the document from EUC-JP to UTF-8 for R and produce the LaTeX document `input-file.tex` in EUC-JP encoding. You can simply compile this tex file with latex command.

If you do not know the supported encoding names, simply put any invalid string for the encoding option, JSweave will display the error message with all the supported encoding names.

To extract R codes from the Rnw document, use the '`-stangle`' option as:

```
$ java -jar {PATH_TO_JAR}/jsweave.jar -stangle input-file.Rnw
```

This will create `input-file.R` in the current directory.

[2]Sweave manual is available from http://www.ci.tuwien.ac.at/~leisch/Sweave.

[3]Note that only SweaveSyntaxNoweb is supported in JSweave. See the Sweave manual for details.

4.1 'type' option

JSweave provides two formats for R code. In addition to the standard format that Sweave provides for R input and output, JSweave introduces a `SOURCE` type. When `type=SOURCE` is set in `<< >>=`, the code chunk is formatted without the R input prompt. If you want the input prompt style, simply do not write the type option at all (alternative value for type option such as '`type=INPUT`' is not implemented yet).

When the source type is set, the R output is always hidden even if the code is evaluated.

Here is an example of the two different types. The following is the JSweave source.

```
Here is the standard Sweave format with input prompt.
JSweave removes gaps between input and output to save spaces.
Note also that the comments are removed by Sweave.
<<>>=
e <- rnorm(n=100, mean=0, sd=20) # generate residuals
x <- 1:100                       # 100 observations
alpha <- 5                       # true intercept
beta <- 1/3                      # true coefficient
y <- alpha + beta*x + e          # generate y
lm(y~x)                          # simple OLS
@
In the following, you can compare the same code with the source type
presentation.  Comments are kept by JSweave and the output is suppressed.
<<type=SOURCE>>=
e <- rnorm(n=100, mean=0, sd=20) # generate residuals
x <- 1:100                       # 100 observations
alpha <- 5                       # true intercept
beta <- 1/3                      # true coefficient
y <- alpha + beta*x + e          # generate y
lm(y~x)                          # simple OLS
@
In source format type, R output is never displayed.  So the option
\texttt{results=SHOW} is simply ignored and teh output is always
supressed.  You can confirm this in the next example.
<<type=SOURCE, results=SHOW>>=
1 + 2
4 + 5
@
```

The above JSweave source will produce something like the following via LaTeX.

Here is the standard Sweave format with input prompt. JSweave removes gaps between input and output to save spaces. Note also that the comments are removed by Sweave.

```
> e <- rnorm(n=100, mean=0, sd=20) # generate residuals
> x <- 1:100                       # 100 observations
> alpha <- 5                       # true intercept
> beta <- 1/3                      # true coefficient
> y <- alpha + beta*x + e          # generate y
> lm(y~x)                          # simple OLS
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     3.1413       0.3763
```

In the following, you can compare the same code with the source type presentation. Comments are kept by JSweave and the output is suppressed.

```
e <- rnorm(n=100, mean=0, sd=20) # generate residuals
x <- 1:100                       # 100 observations
alpha <- 5                       # true intercept
beta <- 1/3                      # true coefficient
y <- alpha + beta*x + e          # generate y
lm(y~x)                          # simple OLS
```

In source format type, R output is never displayed. So the option `results=SHOW` is simply ignored and teh output is always supressed. You can confirm this in the next example.

```
1 + 2
4 + 5
```

4.2 'expand' option

You can control the expansion of reused code in JSweave. Set `expand=FALSE` when you do not want to expand the referring code. Unless `eval=FALSE` is set the code is evaluated even if expand option is set false. When `expand=TRUE` is set in `<< >>=`, the reused code chunk is expanded recursively. The reused code chunk does not have to be defined before it is referred.

```
The following code refers to the `addition' code chunk and the
`addition' in turn refers to the `inner chunk'.  The expand option
expands the referring code recursively.
<<expand=TRUE>>=
a <- 1
b <- 2
<<addition>>
@
Here is the definition of `addition' that also refers
to the `inner chunk' which is not expanded since the expand option is
set `false'.
<<addition, expand=FALSE>>=
y <- a + b
<<inner chunk>>
@
The definition of `inner chunk' is as follows:
<<inner chunk>>=
y
a * 2 + b
@
```

This will produce the following in LaTeX format.

The following code refers to the ‘addition’ code chunk and the ‘addition’ in turn refers to the ‘inner chunk’. The expand option expands the referring code recursively.

```
> a <- 1
> b <- 2
> y <- a + b
> y
[1] 3
> a * 2 + b
[1] 4
>
```

Here is the definition of ‘addition’ that also refers to the ‘inner chunk’ which is not expanded since the expand option is set ‘false’.

```
<<addition>>
```

```
> y <- a + b
> # <<inner chunk>>
```

The definition of ‘inner chunk’ is as follows:

```
<<inner chunk>>
```

```
> y
[1] 3
> a * 2 + b
[1] 4
```

As you can see, the non-expanded code chunk is commented out at the input prompt. If you set the type option to `SOURCE`, this is not the case. The next example modifies the above example only for the last two code chunks set as source type.

```
The following code refers to the `addition' code chunk and the
`addition' in turn refers to the `inner
chunk'.  The expand option expands the refering code recursively.
<<expand=TRUE>>=
a <- 1
b <- 2
<<addition>>
@
Here is the definition for `addition' that also refers
to the `inner chunk' that is not expanded since the expand
option is set `false'.
<<addition, expand=FALSE, type=SOURCE>>=
y <- a + b
<<inner chunk>>
@
```

```
The definition of `inner chunk' is as follows:
<<inner chunk, type=SOURCE>>=
y
a * 2 + b
@
```

This will produce the following in LaTeX format.

The following code refers to the 'addition' code chunk and the 'addition' in turn refers to the 'inner chunk'. The expand option expands the refering code recursively.

```
> a <- 1
> b <- 2
> y <- a + b
> y
[1] 3
> a * 2 + b
[1] 4
>
```

Here is the definition for 'addition' that also refers to the 'inner chunk' that is not expanded since the expand option is set 'false'.

```
<<addition>>
    y <- a + b
    <<inner chunk>>
```

The definition of 'inner chunk' is as follows:

```
<<inner chunk>>
    y
    a * 2 + b
```

Note in Sweave the above documents cause an error since the `<<addition>>` code chunk is used before it is defined. Note also that the code part is indented in JSweave. This is to make it easier to distinguish between the name of the chunk and the code reusing.

The default value for the '`expand`' option is `FALSE` for the `SOURCE` type. However, for the non-source type (with R input prompt and output), the default value for '`expand`' is `TRUE`. See the following example.

```
First, let's consider the default for the `expand' option for the console
type.  The following code should all be expanded by default.
<<console type>>=
<<preparation>>
y <- a + b
@
```

```
However, the source type does not expand the referring code by default
so the following source type code should refer to the `preparation'
code chunk.
<<source type, type=SOURCE>>=
<<preparation>>
y <- a + b
@
The reason why the reused code is expanded for the `console' type and
not for the `source' type is because readers usually expect exactly
the same output as the console in their display, thus it is expanded
by default.

Now, let us suppress expansion for the console type.  Since it is
opposite to the default, you have to explicitly set the expand option false.
<<console type, expand=FALSE>>=
<<preparation>>
y <- a + b
@
The following example sets the expansion option true for the source
type.
<<source type, type=SOURCE, expand=TRUE>>=
<<preparation>>
y <- a + b
@
Finally, here is the definition for \verb|<<preparation>>|
used above.
<<preparation>>=
a <- 1
b <- 2
@
```

This will produce the following in LaTeX format.

First, let's consider the default for the 'expand' option for the console type. The following code should all be expanded by default.

```
> a <- 1
> b <- 2
> y <- a + b
```

However, the source type does not expand the referring code by default so the following source type code should refer to the 'preparation' code chunk.

```
<<preparation>>
y <- a + b
```

The reason why the reused code is expanded for the 'console' type and not for the 'source' type is because readers usually expect exactly the same output as the console in their display, thus it is expanded by default.
Now, let us suppress expansion for the console type. Since it is opposite to the default, you have to explicitly set the expand option false.

```
> # <<preparation>>
> y <- a + b
```

The following example sets the expansion option true for the source type.

```
a <- 1
b <- 2
y <- a + b
```

Finally, here is the definition for `<<preparation>>` used above.

`<<preparation>>`

```
> a <- 1
> b <- 2
```

Note that only the referred code label is printed.

5 Reference

5.1 Default value for options

Newly introduced options in JSweave

1. `type=SOURCE`
2. `expand={FALSE, TRUE}`
   `FALSE` is the default for `SOURCE` type, `TRUE` for console type.

The other options' default values are all the same as those in Sweave except when `type=SOURCE` is set. When the `SOURCE` type is used, the behavior of the following options are changed.

1. `results` option is ignored since the `SOURCE` type is only to display the source without the interaction with R.
2. `echo` option is irrelavent since `type=SOURCE` means always to echo the source code without R prompt.
3. Only '`eval`' options are effective. Other options are ignored.

5.2 List of command options

-stangle extracts the R code from the JSweave document.

-encoding sets the encoding name of the JSweave document and the output document for latex.