John Miyamoto, Department of Psychology, Box 351525
University of Washington, Seattle, WA 98195

Documentation for JM R Utility Functions
Last Updated: Spring 2012

Contents **(Cntrl-left click on a link to jump to the corresponding section)**

1. Preliminary Setup TOC

All of the examples shown below assume that the file, **jm.utilities.rda**, has been attached to the R search path. The following steps will accomplish this purpose.

- Go to http://faculty.washington.edu/jmiyamot/downloads.htm and download **jm.utilities.rda**.
- Save **jm.utilities.rda** to any convenient directory.
- Run R. Then give the R commands:

Table 1. Attach **jm.utilities.rda** to the search path

# R Code	# Explanation

`# "location" denotes the directory that contains jm.utilities.rda.`
 	
`attach("c:/location/jm.utilities.rda")`
	Attach **jm.utilities.rda** to the search path.

`search()`	The **search** command should show that **jm.utilities.rda** is on the search path.

The functions in **jm.utilities.rda** are a subset of John Miyamoto's personal set of functions, which are contained in another file, **jmfuns.rda**.

2. Function Documentation

3. `attach.jm` function documentation TOC

Description: **`attach.jm`** attaches a dataframe or file to the search path in position = pos. **`attach.jm`** is like **`attach`** except that it checks whether the file or dataframe to-be-attached is already attached to the search path.

Usage:

```
attach.jm <- function( filename, pos = 2, allow.2same = FALSE,
    refresh.path = FALSE, case.sensitive = FALSE, name.on.path = character(0),
    abbrev.path = TRUE, feedback = TRUE, infer.DriveLetter = TRUE)
```

Arguments:

`filename`	File to attach to the search path.
`pos`	
`allow.2same`	If **`allow.2same = TRUE`**, then filename will be attached to the search path whether or not another version is already attached. The default is **`allow.2same = FALSE`**.
`refresh.path`	If **`refresh.path = TRUE`**, & **`filename`** is already attached to the search path, then all older versions of filename are detached from the search path, and a new fresh version is attached to the search path. If **`refresh.path = FALSE & allow.2same = FALSE`**, then **`filename`** is attached to the search path if and only if no same-name object or file already exists on the search path. The default is **`refresh.path = FALSE`**.
`case.sensitive`	
`name.on.path`	The **`name.on.path`** argument has no effect if **`filename`** is the name of a file. If **`filename`** is a dataframe or matrix and **`name.on.path != character(0)`**, **`filename`** will be attached to the search path under the name, **`name.on.path`** provided that either there is no object named **`name.on.path`** that is already on the search path or **`refresh.path = TRUE`** or **`allow.2same = TRUE`**. If **`filename`** is a dataframe or matrix, and **`name.on.path = character(0)`** (default), then the dataframe or matrix will be attached with its current name as the name of the environment on the search path.
`abbrev.path`	
`feedback`	If **`feedback = TRUE`** (default), the function displays feedback re the current status of the search path. If **`feedback = FALSE`**, this feedback is not displayed.
`infer.DriveLetter`	If **`infer.DriveLetter = TRUE`** (default), two file specifications will be treated as identical if one of them lacks a drive letter, e.g., **`/aa/bb/file.rda`** will be treated as the same file as **`c:/aa/bb/file.rda`** if **`getwd()`** returns **`c:`** as the drive for the current working directory. It has no effect if the root directory is not specified, e.g., **`file.rda`** and **`c:/aa/bb/file.rda`** will be treated as different even if the current working directory id **`c:/aa/bb`**. Similarly, **`c:/aa/file.rda`** and **`e:/aa/file.rda`** are treated as different files.

Table 2. Examples of attach.jm

# R Code	# Explanation
`# If there exists a function called` **`# "c:/mydata/myfuns.rda"`** **`attach.jm( "c:/mydata/myfuns.rda" )`**	Attaches **`"c:/mydata/myfuns.rda"`** to the search path.
`attach.jm( "c:/mydata/myfuns.rda" )`	Refuses to attach **`"c:/mydata/myfuns.rda"`** a second time.
`attach( "c:/mydata/myfuns.rda" )`	Note that the standard **`attach`** function of R

# R Code	# Explanation
`search()`	does allow a file to be attached twice to the search path.
`det.jm( "myfuns.rda", all.matches = TRUE )`	Detaches both copies of **"file:c:/mydata/myfuns.rda"** from the search path.

4. `colors.jm` function documentation TOC

5. `det.jm` function documentation TOC

Description: **`det.jm`** detaches objects from the search path and returns nice feedback about the resulting search path. The to-be-detached object is specified by the **`file.spec`** argument; it can be specified as a position (1, 2, 3, ...) or as the name of an object, e.g., **`jmfuns.rda`**. Named objects can be partially matched so that **`jmfuns`** and **`jmfuns.rda`** will identify the same object so long as no other object on the search path matches **`jmfuns`**.

Usage:

```
det.jm(file.spec, first = FALSE, all.matches = FALSE,
   feedback=TRUE, ignore.case = TRUE, nchar.path.names = 35,
   max.path = 15)
```

Arguments:

`file.spec`	Quoted character string that identifies the file to be detached from the search path. **`det.jm`** uses partial matching to identify a file name; thus, **`"file:C:/mydata/jm.utilities.rda"`**, **`"jm.utilities.rda"`**, and **`"jm.util"`** all identify the same file, provided that only one file on the search file has a name that matches the **`file.spec`** string.
`first`	If **`first = FALSE`** (default) and **`all.matches = TRUE`**, then all matching objects on the search path are detached. If **`first = FALSE`** (default) and **`all.matches = FALSE`** (default), then an error message is generated if there are multiple matches, and no objects on the search path are detached. If **`first = TRUE`**, **`det.jm`** detaches the first instance of **`file.spec`** on the search path without warning the user that multiple matches were found.
`all.matches`	If **`all.matches = TRUE`**, all files on the search path that match **`file.spec`** are detached from the search path. If **`all.matches = FALSE`**, then the file with the uniquely matching file name is detached if it exists; if multiple matches are found, no file is detached and an error message is returned.
`feedback`	**`feedback = TRUE`** (default) requests feedback regarding the result of the detachment. If **`feedback = TRUE`**, then **`nchar.path.names`** determines the number of characters that are displayed from any name on the search path. Names that are longer are shortened to this length. If **`feedback = FALSE`**, the function returns no information about the detachment operation.
`ignore.case`	If **`ignore.case = FALSE`**, the **`file.spec`** is case-insensitive. If **`ignore.case = TRUE`**, the **`file.spec`** is case-sensitive. The default is **`ignore.case = FALSE`**.
`nchar.path.names`	**`nchar.path.names`** and **`max.path`** are used to control the display of feedback on the screen. They do not affect the detachment operation.
`max.path`	**`max.path`** specifies the maximum number of path positions displayed.

Table 3. Examples of the det.jm function

# R Code	# Explanation

`# If there exists a function called`
`# "c:/mydata/myfuns.rda"`
 	
`attach.jm( "c:/mydata/myfuns.rda" )`
	Attaches **"c:/mydata/myfuns.rda"** to the search path.

# R Code	# Explanation

`search()`
 	
`det.jm( "myfuns.rda" )`
 	
`search()`	

`attach( "c:/mydata/myfuns.rda" )`
`attach( "c:/mydata/myfuns.rda" )`
 	
`search()`
	Note that the **attach** function in R "**c:/mydata/myfuns.rda**" to the search path twice.

`det.jm( "myfuns.rda", all.matches = TRUE )`	Detaches both copies of **"file:c:/mydata/myfuns.rda"** from the search path. Note also that **det.jm** makes use of partial matching, so that "**myfuns.rda**" suffices to identify the file to be detached as long as it is unique on the search path.

6. doc function documentation TOC

Description: **doc(x)** prints documentation for **x** to the screen. Documentation is stored in the **doc** attribute of **x** (old method) or in a separate **x.doc** object (current method).

Usage:

```
doc(x, suppress.o.type = FALSE, widthX = 1.0)
```

Arguments:

x	**x** must be an object with a **doc** attribute or else there must exist a **x.doc** object. Optionally **x** can be the name of a string, e.g., x = **aaa.doc** where **aaa.doc** is a character string (typically documentation of some kind).
suppress.o.type	Set **suppress.o.type = TRUE** to suppress the display of the object type (**o.type** function output).
widthX	**widthX** can be used to alter the proportion of the screen width that is covered by a line of text before it begins to wrap. The default is **widthX = 1.0**.

```
123456789 123456789 123456789 123456789 123456789 123456789 123456789 123456789 1234
```

Table 4. Examples of the **doc** function

# R Code	# Explanation

`xx = c(1, 3, 5)`
 	
`xx.doc = "`
`xx is just an example of an R object.  This text is sample documentation for xx.`
 	
`This is paragraph 2 of the documentation.  Blah, blah, blah.`
 	
`End of Documentation.`
`"`
	Create an object **xx** and documentation for **xx**.

`doc(xx)`	Displays the documentation for **xx**.
`doc(xx, widthX = .8)`	Displays the documentation for **xx** with narrower paragraphs.
`doc( doc )`	Displays the documentation for the **doc**

# R Code	# Explanation
	function.

7. `ls.jm` function documentation TOC

Description: **`ls.jm`** outputs a vector of object names for objects in an environment on the search path. It is essentially the same as **`ls`** except that the formatting is easier to work with.

Usage:

```
ls.jm(x = 1, ncol = NA, display.std = 85, ...)
```

Arguments:

`x`	**`x`** is the position in the search path whose objects are to be displayed. **`x`** can be set either as a digit or as the character name of the environment. **`x = 1`** designates the **`.GlobalEnv`**. Names of files are identified by partial matching, e.g., if **`"file:c:/mydata/jm.utilities.rda"`** is on the search path, then **`ls.jm("jm.utilities")`** will display the objects in this file.
`ncol`	Number of columns to be used in the display.
`display.std`	The assumed screen width in the display.
`...`	Other arguments to be passed to **`ls`**.

Table 5. Examples of **`ls.jm`**

# R Code	# Explanation
`search()`	Find location of **`jm.utilities.rda`** on the search path. If it is not on the search path, you will have to attach it in order to reproduce the following examples.
`ls.jm( "jm.utilities.rda" )`	Displays the objects in **`jm.utilities.rda`**.
`ls.jm( "jm.util" )`	Note that both the previous and current use of **`ls.jm`** uses partial matching to identify a file on the search path, provided that it is unique.
`search()` `ls.jm( "a" )`	An error message is produced if the inputted string matches more than one file on the search path.

8. `make.names.jm` function documentation TOC

Description: **`make.names.jm`** makes syntactically valid R names out of a character vector (as does the **`make.names`** function). It differs from **`make.names`** insofar as terminal periods (.) are deleted if possible, and successive periods, e.g., **`".."`** is optionally reduced to **`"."`**. These aspects can be useful when formatting the names of variables that were outputted by OpenBUGS or JAGS.

Usage:

```
make.names.jm(names.jm, unique.jm = TRUE, rm.double = TRUE, allow_.jm = TRUE)
```

Arguments:

`names.jm`	**`names.jm`** is a character vector to be turned into valid R object names.
`unique.jm`	**`unique.jm = TRUE`** (default) guarantees that the names are unique.
`rm.double`	**`rm.double = TRUE`** (default) reduces repetitions of periods to single periods, e.g., **`x..y`** is converted to **`x.y`**.
`allow_.jm`	**`allow_.jm = TRUE`** (default) allows underscores in the names.

Table 6. Examples of **make.names.jm**

# R Code	# Explanation

`make.names.jm(`
`  c("f[1]","f[2]","x[[3]]","p[[k(n)]]"),`
`  rm.double = F)`	

`make.names.jm(`
`  c("f[1]","f[2]","x[[3]]","p[[k(n)]]"),`
`  rm.double = T)`	

`make.names.jm( c("beta[1]", "beta[15]", "beta[21]") )`	
`make.names.jm( c("tau[1,1]", "tau[1,2]",` `  "tau[2,1]", "tau[2,2]" ) )`	

9. move function documentation TOC

Description: The function move moves an object from one environment on the search path (source) to another environment (destination) on the search path. The default operation deletes the object from the source environment after moving it; it also saves the destination environment to its corresponding file. The typical use of move moves an object from .GlobalEnv to a file that is attached to the search path. By default, the move is not carried out if an object with the same name exists in the destination directory. The default can be overridden (replace=T). The move function also applies to a character vector of object names.

Usage:

```
move(x, to, from = ".GlobalEnv", replace.objects = FALSE, move.doc.objects = TRUE,
    copy.only = FALSE, map = T)
```

Arguments

x	Object to be moved; or a character vector of object names which will be moved.
to	The file to which the objects are to be move. to can be specified as either a position, e.g., to = 3 specifies the 3rd position on the search path, or as a named environment, e.g., to = "data.rda". Partial matching is used to match the named environment if an incomplete name is given. Thus, to = "data.rda" matches file:c:/myproject/data.rda provided that it is the only file on the search path that matches data.rda.
from	The source file or environment from which the objects are to be moved. from can be specified as either a position, e.g., from = 3, or as a named environment, e.g., to = "data.rda". Partial matching is used to match the named environment if an incomplete name is given.
replace.objects	replace.objects = TRUE to replace a same name object in destination directory. If replace.objects = FALSE (default), then no move is carried out when a same name object exists in the destination file.
move.doc.objects	move.doc.objects = TRUE causes XXX.doc objects to be moved along with the XXX object. If FALSE, only the XXX object is moved. (See Section 6 for an explantion of .doc objects.)
copy.only	If copy.only = TRUE, then the object is copied to the destination file, but is not deleted from the source environment. If it is FALSE, then the object is deleted after copying it to the destination file. The default if copy.only = FALSE.
feedback	If feedback = TRUE (default), then feedback is printed to the screen regarding which objects have been successfully moved and which moves have failed. If feedback = FALSE, this feedback is suppressed.
map	If map = TRUE, a table showing the existence and replacement of objects is displayed; if FALSE, no table is displayed. The default is map = TRUE.

Table 7: Examples of the **move** function. The examples assume that **jm.utilities.rda** has been attached to the search path.

# R Code	Explanation

`aa = 1:3`
`bb = 11:13`
`save(aa, bb, file = "temp1.rda")`
`rm(list = c("aa", "bb"))`
	Create a file called, **temp1.rda** with two objects in it.

`attach("temp1.rda", pos = 2)`	Attach "**temp1.rda**" to the search path.

`search()`
 	
`ls.jm( "temp1.rda" )`
	Note that **temp1.rda** is now attached to the search path.

`cc = list(`
`  x = "Label for this list",`
`  just.a.matrix = matrix(1:12, ncol = 3) )`
 	
`cc`
	Now we create some new objects. They will be moved into temp1.rda after checking the contents of **.GlobalEnv** and **temp1.rda**.

`ls.jm(1)`
`ls.jm(2)`
`move(list = cc, "temp1")`
`ls.jm(1)`
`ls.jm(2)`
	Note the contents of **.GlobalEnv** (position 1) and **temp1.rda** (position 2) before and after the **move**. The **move** command moves **cc** to **temp1.rda**. Note that the **move** command recognizes the search path position of **temp1.rda** based on partial matching of the file name. If there were multiple files on the search path that all matched **temp1.rda**, e.g., **aa.temp1.rda** and **bb.temp1.rda**, then no **move** would be carried out.

`dd = (1:3)*2`
`ls(1)`
`ls(2)`
`move(dd, 2)`
`ls(1)`
`ls(2)`
	This code is similar to the preceding code except that the to argument is specified by the numeric position, 2, rather than by a partial name, **temp1**.

`ff = function(x) { x^2 }`
 	
`ff.doc = c(`
`"The ff function computes the square of its input.")`
 	
`doc(ff)`
	Next: Create a function with documentation for the function.

`ls(1)`
`ls(2)`
`move(ff, "temp1")`
`ls(1)`
`ls(2)`
	Note that **move** automatically moves the corresponding doc object, i.e., by moving **ff**, it also moves **ff.doc**. This default can be suppressed.

`ls(2)`
`aa = "new version of aa"`
`bb = "new version of bb"`
`move(c("aa", "bb"), "temp1")`
`ls(2)`
	Note that **move** by default will not replace existing objects.

`ls(1)`
`move(c("aa", "bb"), "temp1", rep = TRUE)`
`ls(2)`
	To replace existing objects, set **`replace.objects = TRUE`**).

`xx = 1:3`
`save(xx, file = "temp2.rda")`
`attach("temp2.rda", pos = 3)`
`search()`
	Create another R file and attach it to the search path.

`ls(2)`
`ls(3)`
`move(c("aa", "bb"), to = "temp2", from = "temp1")`
`ls(2)`
`ls(3)`
	Move some objects from **`temp.1.rda`** to **`temp2.rda`**. Note that the to and from arguments identify files by partial matching of names.

`ls.jm(2)` `ls.jm(3)`	The **`ls.jm`** function is like the **`ls`** function except that the output is formatted differently. See Section 7 for a description of the **`ls.jm`** function
`det.jm("temp1")` `det.jm("temp2")`	The **`det.jm`** function is like the detach function except that environments on the search path can be matched by partial matching of the names and the output gives better feedback regarding the result of the detachment. See Section 5 for a description of the det.jm function.

10. `o.type` function documentation TOC

Description: **`o.type`** tests for the mode, factor status and other classifications of an object. The information about the object is displayed in the screen.

Usage:

```
o.type(x, variables = FALSE, sorted = TRUE)
```

Arguments:

`x`	**`x`** is the object to be tested. Its characteristics are displayed on the screen.
`variables`	**`variables = TRUE`** only has an effect if **`x`** is a dataframe. In this case, **`variables = TRUE`** causes **`o.type`** to list the information about every variable in **`x`**.
`sorted`	**`sorted = TRUE`** causes the output to printed with the TRUE attributes first; otherwise the attributes are always printed in the same order. **`sorted = TRUE`** has no effect if **`x`** is a dataframe and **`variables = TRUE`**.

Possible Object Types:

`all.NA` = all elements are NA;
`array` = is an array;
`char` = character vector or matrix;
`d.frame` = is a dataframe;
`factor` = is a factor;
`fn` = is a function;
`list` = is a list;
`logical` = is logical;
`matrix` = is a matrix;

NULL = is null;

numeric = is numeric;

scalar = is a scalar;

zero = has length zero.

Note that a factor is false for **is.numeric** even though its mode is numeric (R-oddity). Moreoever factors and string variables are false for 'is.vector' if they have any attributes other than names (which they typically have).

Table 8. Examples that use the **o.type** function

# R Code	# Explanation
aa = c(1,3,5) aa o.type(aa)	Check the object type of **aa**.
bb = matrix(1:12, ncol = 4) bb o.type(bb)	Check the object type of **bb**.
cc = array(1:16, dim = c(2, 4, 2)) cc o.type(cc)	Check the object type of **cc**.
pp = list(aa = aa, bb = bb, cc = cc) pp o.type(pp)	Check the object type of **pp**.
xx = 1:5 yy = 1.5 * xx + rnorm(5, 0, 1) lm.out = lm(yy ~ xx) lm.out names(lm.out) o.type(lm.out)	Check the object type of **lm.out**. Note that **lm.out** is a list that is a special class.
fac1 = c(1, 2, 2, 1, 2, 1, 1, 1, 2) fac2 = c(1, 3, 2, 3, 1, 3, 3, 2, 1) (tbl = table(fac1, fac2)) o.type(tbl)	Check the object type of **tbl**. Again, **tbl** is a special class of matrix.
ff = factor(c("aa", "bb", "aa")) ff o.type(ff)	Check the object type of **ff**.
aa = c(1,3,5) ff = factor(c("aa", "bb", "aa")) kk = c(TRUE, FALSE, TRUE) DD = data.frame(aa, ff, kk) DD	Create a dataframe to be checked for object type.
o.type(DD)	Checks the object type of **DD**.
o.type(DD, variable = TRUE)	Checks the object types of the variables in **DD**.

11. pchlist function documentation TOC

Description: Give the command, **pchlist()**, to see a display of the different plotting symbols and their codes.

Usage:

```
pchlist()
```

Arguments: None

Table 9. Example of **pchlist**

# R Code	# Explanation
`pchlist()`	Use **pchlist()** to choose plotting symbols.

`xx = c( 2, 3, 2.5, 4, 2.5, 3.3 )`
`yy = c( 5, 10, 9, 12, 7.3, 9 )`
 	
`plot( xx, yy, pch = 1, cex = 2 )`	

`pchlist()`	Use **pchlist()** to choose plotting symbols.
`plot( xx, yy, pch = 19, cex = 2 )`	
`pchlist()`	Use **pchlist()** to choose plotting symbols.
`plot( xx, yy, pch = 4, cex = 2 )`	

12. `plot.jm` function documentation TOC

Description: **plot.jm** makes the JM default plot.

Usage:

```
plot.jm(x=c(0,100), y=c(0,100), no.margins = FALSE, type= "n", ...)
```

Arguments:

x	The lower and upper limits on the X axis. The default is **x = c(0,100)**.
y	The lower and upper limits on the Y axis. The default is **y = c(0,100)**.
no.margins	If **no.margins = TRUE**, the plotting area has no space for margins. This is preferred when making text plots or diagrams. The default value is **no.margin = FALSE**. If **no.margins = FALSE**, the plotting area is created with the default margins.
type	The plot type. The setting, **type = "n"**, suppresses plotting (to be finished by subsequent calls to **lines** and **points**. See below for other plot types.
...	Other arguments to be passed to **plot**.

13. `rm.sv` function documentation TOC

Description: The function **rm.sv** deletes an R object from another environment on the search path, and then saves that environment to its associated file. The rm.sv function also applies to the objects specified by a character vector of object names.

Usage:

```
rm.sv( list, env.on.path = "", pos = NA, rm.doc.objects = TRUE,
          feedback = TRUE )
```

Arguments:

list	The object or objects to be deleted; to specify multiple objects, specify the objects with a character vector of object names.
env.on.path	**env.on.path** = the name of R-environment containing the object; Specify **pos** or **env.on.path**, but not both. Note that **env.on.path** has the form, **env.on.path = "c:/..." or** env.on.path = "**c:\\...**", and NOT as, **env.on.path = "file:c:\..."**, i.e., directories on a path are separated by a forward slash or a double backslash and not by a single backslash. Furthermore, **env.on.path** is

	matched by partial matching, so that **env.on.path = "myfuns.rda"** will be matched to **"file:c:/job/myfuns.rda\"** if it is on the search path (and a unique match). Note also that **env.on.path** and **pos** may NOT specify a package. If **env.on.path = ""** (default), then the environment is set by **pos**.
pos	**pos** = position number of the directory containing the object; by default, pos = NA. The environment where the object is to be deleted should be specified by **env.on.path** or by **pos**, but not both.
rm.doc.objects	If **rm.doc.objects = TRUE**, then .doc objects that correspond to objects in **list** will also be deleted. If **rm.doc.objects = FALSE**, then only the objects names in **list** will be deleted.
feedback	**feedback = TRUE** (default) means that the call to **rm.sv** will give feedback about the success or failure of the move. **feedback = FALSE** omits this feedback.

Table 10. Examples of the **rm.sv** function

# R Code	# Explanation

`aa = 1:3`
`bb = 11:13`
`save( aa, bb, file = "temp1.rda" )`
`rm(list = c("aa", "bb"))`
	Create a file called, **temp1.rda** with two objects in it.

`attach( "temp1.rda", pos = 2 )` `ls.jm( "temp1.rda" )`	Attach "**temp1.rda**" to the search path.

`search()`
 	
`ls.jm( "temp1.rda" )`
	Note that **temp1.rda** is now attached to the search path and it contains objects **aa** and **bb**.

`rm.sv(aa, "temp1.rda")`
 	
`ls.jm( "temp1.rda" )`
	Remove **aa** from **temp1.rda** on the search path, and then save **temp1.rda** to its associated file (happens automatically within the **rm.sv** function.

`cc = list(`
`  x = "Label for this list",`
`  just.a.matrix = matrix(1:12, ncol = 3) )`
`dd = (1:3)*2`
 	
`cc`
`dd`
`move( list = c( "cc", "dd" ), "temp1.rda")`
`ls.jm( "temp1.rda" )`
	Now we create some new objects. They will be moved into temp1.rda after checking the contents of **.GlobalEnv** and **temp1.rda**.

`rm.sv( list = c( "bb", "cc" ), "temp1.rda" )`
 	
`ls.jm( "temp1.rda" )`
	We can remove more than one object from an environment on the search path, but we must pass the names of the to-be-removed objects to **rm.sv** as a character vector of object names.

`ff = function(x) { x^2 }`
 	
`ff.doc = c(`
`"The ff function computes the square of its input.")`
 	
`doc(ff)`
	Next: Create a function with documentation for the function.

`ls(1)` `ls(2)`	Note that **move** automatically moves the corresponding doc object, i.e., by moving **ff**,

# R Code	# Explanation
move(ff, "temp1") **ls(1)** **ls(2)**	it also moves **ff.doc**. This default can be suppressed.
rm.sv(ff, "temp1.rda") **ls.jm("temp1")**	Note that if we remove an object that has corresponding **doc** object, e.g., **ff** and **ff.doc**, then removing the object removes the corresponding doc object.
det.jm("temp1")	The **det.jm** function is like the detach function except that environments on the search path can be matched by partial matching of the names and the output gives better feedback regarding the result of the detachment. See Section 5 for a description of the det.jm function.

14. `save.jm` function documentation TOC

Description: **save.jm** is exactly like the R **save** function except that it checks whether the target file exists. If it does, then it won't save the objects to this file unless specifically instructed to do so.

Usage:

```
save.jm( list = character(0), overwrite = FALSE,
   file = stop("'file' must be specified."), ascii = FALSE,
   version = NULL, envir = parent.frame(), compress = FALSE )
```

Arguments:

list	**list** is a vector of object names.
overwrite	If **overwrite = FALSE** (default), then **save.jm** will not overwrite an existing file. If the user attempts to write to an existing file, the function call stops with an error message that states that the target file exists. If **overwrite = TRUE**, then **save.jm** will overwrite an existing file if the target file already exists.
The remaining arguments of **save.jm** are taken directly from the arguments of **save**. R-help gives the following description of the arguments:	
file	**file** is a (writable binary-mode) connection or the name of the file where the data will be saved (when tilde expansion is done). Must be a file name for **version = 1**.
ascii	**ascii:** If **TRUE**, an ASCII representation of the data is written. The default value of **ascii** is FALSE which leads to a binary file being written.
version	**version** indicates the workspace format version to use. **NULL** specifies the current default format. The version used from R 0.99.0 to R 1.3.1 was version 1. The default format as from R 1.4.0 is version 2.
envir	**envir** indicates the environment to search for objects to be saved.
compress	**compress** is a logical or character string specifying whether saving to a named file is to use compression. **TRUE** corresponds to gzip compression, and (from R 2.10.0) character strings "gzip", "bzip2" or "xz" specify the type of compression. Ignored when file is a connection and for workspace format version 1.

15. `setwd.jm` function documentation TOC

Description: **setwd.jm** sets the current working directory and attaches the **data.rda** file in this directory, if one exists. If no such file exists, then it creates such a file.

Usage:

```
setwd.jm(dir2)
```

Arguments:

dir2	**dir2** names the new working directory.