g.data Package Documentation

David Brahm

December 16, 2013

Abstract

Normally in R, objects live – and die – in memory unless you explicitly save them with `save`, or save the entire image with `save.image`. The `g.data` package allows you to save a whole group of objects to an associated directory on disk, then access them later. The objects then appear to exist in a particular location on the search path (position 2 by default), and are readily accessible without extra effort, but R does not actually load them into memory until needed.

1 Introduction

In this example, I create two large matrices `m1` and `m2`, and store them on disk in a "delayed data package" (ddp). Normally you'd choose the ddp location, but here it's just a temporary directory. The `g.data.attach` command attaches an environment associated with the ddp directory:

```
> require(g.data)
> (ddp <- tempfile("newdir"))                    # Where to put the files

[1] "/tmp/RtmpqTcrFi/newdir2e37173a128"

> g.data.attach(ddp)                  # Warns that this is a new directory
> search()[1:3]

[1] ".GlobalEnv"        "newdir2e37173a128" "package:g.data"

> assign("m1", matrix(1, 5000, 1000), 2)
> assign("m2", matrix(2, 5000, 1000), 2)
> ls(2)

[1] "m1" "m2"
```

The `g.data.save` command does the actual storing to disk. Once I detach the environment they lived in, R forgets the objects:

```
> g.data.save()                                    # Writes the files
> detach(2)
```

In the same or another R session, I then attach the ddp, and the matrices appear to be instantly accessible. In fact they are just promises, so the first time I access `m1` (by asking its dimensionality) there is a delay as `m1` is actually loaded into memory. Further access to `m1` is quick, though, because now it's in memory. Note `m2` never needs to be loaded into memory, saving time and resources:

```
> g.data.attach(ddp)                # No warning, because directory exists
> ls(2)

[1] "m1" "m2"

> system.time(print(dim(m1)))                    # Takes time to load up

[1] 5000 1000
   user  system elapsed
  0.058   0.005   0.064

> system.time(print(dim(m1)))                   # Second time is faster!

[1] 5000 1000
   user  system elapsed
  0.000   0.000   0.001

> find("m1")                          # m1 still lives in pos=2, is now real

[1] "newdir2e37173a128"
```

I can also put a new object `m3` into the ddp and re-save it:

```
> assign("m3", m1*10, 2)
> g.data.save()                              # Or just g.data.save(obj="m3")
> detach(2)
```

2 Variations

There is a function `g.data.get` to access a single object without attaching the ddp:

```
> mym2 <- g.data.get("m2", ddp)          # Get one object without attaching
```

There is also a function `g.data.put` to write an object without attaching the ddp:

```
> g.data.put("m4", matrix(1:12, 3,4), ddp)
```

Since we're done with this example, you may want to remove the ddp now:

```
> unlink(ddp, recursive=TRUE)                    # Clean up this example
```

Here is a new example with a slightly different approach. We skip `g.data.attach` entirely, instead attaching a list `y` directly to position 2. `g.data.save` still works, but you must now tell it the location of the directory:

```
> ddp <- tempfile("newdir")
> y <- list(m1=1:1000, m2=2:1001)
> attach(y)                          # Attach an existing list or dataframe
> search()[1:3]

[1] ".GlobalEnv"     "y"              "package:g.data"

> ls(2)

[1] "m1" "m2"

> g.data.save(ddp)
> detach(2)
> unlink(ddp, recursive=TRUE)                    # Clean up this example
```

3 Under the Hood

`g.data.save` simply stores one object per file in the ddp directory. An object `xyz` is stored in file `xyz.RData`. You could access these files with ordinary `load` commands, and you could write (or overwrite) them with `save` commands.

Unfortunately, in Windows the files `x.RData` and `X.RData` are indistinguishable, so we modify the naming convention by preceding uppercase letters with the @ symbol. An object `aBcD` is stored in file `a@Bc@D.RData`.

`g.data.attach` contains the magic. The environment it attaches contains only promises, implemented with `delayedAssign`. When you first access an object, R fulfills the promise to 1) load the data file, 2) store the real object in the environment, and 3) return its value to you. Subsequent access just returns the real object which is now stored in the environment. `g.data.attach` also gives the environment a "path" attribute, so `g.data.save` will know where to write files.

`g.data.save` is smart enough to only write back to disk objects that are not promises. It also has options to allow you to choose the objects written, remove objects, and set the directory to write to.

A Function Index

- **Create and Maintain Delayed-Data Packages**

  g.data.attach: Attach a delayed-data package (DDP)

  g.data.save: Write a DDP to disk

  g.data.get: Get one object from a DDP on disk

  g.data.put: Write one object to a DDP on disk