Object tracking in R

Tony Plate

December 5, 2010

1 Introduction

The track package sets up a link between R objects in memory and files on disk so that objects are automatically saved to files when they are changed. R objects in files are read in on demand and do not consume memory prior to being referenced. The track package also tracks times when objects are created and modified, and caches some basic characteristics of objects to allow for fast summaries of objects.

Each object is stored in a separate RData file using the standard format as used by `save()`, so that objects can be manually picked out of or added to the track database if needed. The track database is a directory usually named `rdatadir` that contains a RData file for each object and several housekeeping files that are either plain text or RData files.

Tracking works by replacing a tracked variable by an `activeBinding`, which when accessed looks up information in an associated 'tracking environment' and reads or writes the corresponding RData file and/or gets or assigns the variable in the tracking environment. In the default mode of operation, R variables that are accessed are stored in memory for the duration of the top level task (i.e., in one expression evaluated from the prompt.) A callback that is called each time a top-level-task completes does three major things:

- detects newly created or deleted variables, and adds or removes from the tracking database as appropriate, and
- writes changed variables to the database, and
- deletes cached objects from memory.

With the track package, R provides a similar user experience to the old S-PLUS in terms of how it stores variables (the good along with the bad).

1.1 How to use the track package

```
> library(track)
> track.start()

Tracking <env R_GlobalEnv> (writable) using existing directory 'rdatadir'
```

```
> track.stop()

Stopping tracking on <env R_GlobalEnv>
```

2 An example of tracking

Here is a brief example of tracking some variables in the global environment:

```
> library(track)
> track.start()

Tracking <env R_GlobalEnv> (writable) using existing directory 'rdatadir'

> x <- 123
> y <- matrix(1:6, ncol = 2)
> z1 <- list("a", "b", "c")
> z2 <- Sys.time()
> track.summary()

            class    mode extent length size            modified TA TW
x         numeric numeric    [1]      1   48 2010-12-05 00:01:07  0  7
y          matrix numeric  [3x2]      6  232 2010-12-05 00:01:07  0  7
z1           list    list  [[3]]      3  360 2010-12-05 00:01:07  0  7
z2 POSIXct,POSIXt numeric    [1]      1  312 2010-12-05 00:01:07  0  7

> ls(all = TRUE)

[1] ".Last"        ".trackingEnv" "x"            "y"            "z1"           "z2"

> track.stop()

Stopping tracking on <env R_GlobalEnv>

> ls(all = TRUE)

character(0)

> track.start()

Tracking <env R_GlobalEnv> (writable) using existing directory 'rdatadir'

> ls(all = TRUE)

[1] ".Last"        ".trackingEnv" "x"            "y"            "z1"           "z2"

> track.summary()
```

```
            class    mode extent length size            modified TA TW
x         numeric numeric    [1]      1   48 2010-12-05 00:01:07  0  7
y          matrix numeric  [3x2]      6  232 2010-12-05 00:01:07  0  7
z1           list    list  [[3]]      3  360 2010-12-05 00:01:07  0  7
z2 POSIXct,POSIXt numeric    [1]      1  312 2010-12-05 00:01:07  0  7
```

```
> track.stop()
```

```
Stopping tracking on <env R_GlobalEnv>
```

```
> list.files("rdatadir", all = TRUE)
```

```
[1] "."                    ".."                   ".trackingSummary.rda" "_1.rda"
[5] "filemap.txt"          "x.rda"                "y.rda"                "z1.rda"
[9] "z2.rda"
```

There are several points to note:

- The global environment is the default environment for tracking – it is possible to track variables in other environments, but that environment must be supplied as an argument to the track functions.
- By default, newly created or deleted variables are automatically added to or removed from the tracking database. This feature can be disabled by supplying `auto=FALSE` to `track.start()`, or by calling `track.auto(FALSE)`.
- When tracking is stopped, all tracked variables are saved on disk and will be no longer accessible until tracking is started again.
- The objects are stored each in their own file in the tracking dir, in the format used by `save()`/`load()` (RData files).

3 Why use the `track` package

There are four main reasons to use the `track` package:

- conveniently handle many moderately-large objects that would collectively exhaust memory or be inconvenient to manage in files by manually using `save()`, `load()`, and/or `save.image()`.
- have changed or newly created objects saved automatically at the end of each top-level command, which ensures objects are preserved in the event of accidental or abnormal termination of the R session, and which also makes startup and saving much faster when many large objects in the global environment must be loaded or saved.
- keep track of creation and modification times on objects
- get fast summaries of basic characteristics of objects - class, size, dimension, etc.

4 Incremental history

The track package also provides a self-contained incremental history saving function that writes the most recent command to the file `.Rincr_history` at the end of each top-level task, along with a time stamp that does not appear in the interactive history. The standard history functionality (savehistory/loadhistory) in R writes the history only at the end of the session. Thus, if the R session terminates abnormally, history is lost.

To turn on incremental history recording, issue the command

```
> track.history.start()
```

To turn it off, issue the command

```
> track.history.stop()
```

The history is stored in a simple text format with time stamps. It can be viewed in an editor, but be careful not to view it in an editor that locks the file while the R session is active (many editors under Windows lock the file they have open, with the exception of emacs.)

5 Cache-policy plugins

There is an option to control whether tracked objects are cached in memory as well as being stored on disk. By default, objects are cached in memory for the duration of a top-level task, but are flushed from memory at the end of a top-level task. This means that when they are accessed again, they must be read from files. To save time when working with collections of objects that will all fit in memory, turn on caching with and turn off cache-flushing `track.options(cache=TRUE, cachePolicy="none")`, or start tracking with `track.start(..., cache=TRUE, cachePolicy="none")`. A possible future improvement is to allow conditional and/or more intelligent caching of objects. Some data that would be needed for this is already collected in access counts and times that are recorded in the tracking summary.

Along these lines, `track` contains an experimental feature that allows users to supply their own plugin functions that specify cache rules. Currently, the plugin function can specify whether or not an object will be flushed from memory at the end of a top-level command.

Here's an example of a cache plugin function that keeps in memory variables whose names begin with the letter 'x'.

```
plugin <- function(objs, inmem, envname) {
    keep <- regexpr("^x", rownames(objs))>0
    return(keep)
}
```

To use this function, supply it to `track.options()`:

```
track.options(cacheKeepFun=plugin, save=TRUE)
```

The plugin function takes three arguments:

- objs: the object summary dataframe - same as returned by `track.summary()`. The names of the objects are in the rownames of the dataframe.
- inmem: a logical vector with length equal to the number of rows in `objs`. It will have value `TRUE` where the corresponding object is in memory, and `FALSE` otherwise.
- envname: a single string containing the name of the tracking environment, in a form like `<env R_GlobalEnv>`.

The plugin function should return a logical vector the same length as `inmem`, with `TRUE` values where the corresponding objects should be kept in memory.

6 What track is not good for

Tracking is not particularly suitable for storing objects that contain environments, because those environments and their contents will be fully written out in the saved file (in a live R session, environments are references, and there can be multiple references to one environment.) Functions are one of the most common objects that contain environments, which can contain data objects local to the function (e.g., see the examples in the R FAQ in the section "Lexical scoping" under "What are the differences between R and S?" `http://cran.r-project.org/doc/FAQ/R-FAQ.html#Lexical-scoping`). Additionally, the results of some modeling functions contain environments, e.g., `lm` holds several references to the environment that contains the data. When an `lm` object is `save`'ed, the environment containing the data, and all the other objects in that environment, can be saved in the same file. To work with large data objects and modeling functions, consider first creating a tracking database that contains the data objects. Then, in a different R session (which can be running at the same time), use `track.attach` to attach the db of data objects at `pos=2` on the search list. When working in this way, the data objects will only be kept in memory when being used, and modeling functions that record environments in their results can be successful used (though beware of modeling functions that store large amounts of data in their results.) Alternatively, use modeling functions that do not store references to environments. The utility function `show.envs()` from the `track` package will show what environments are referenced within an object (though it is not guaranteed to find them all.)