Package ‘track’

July 23, 2016

Version 1.1.9

Date 2016-07-20

Title Store Objects on Disk Automatically

Author Tony Plate <tplate@acm.org>

Maintainer Tony Plate <tplate@acm.org>

Description Automatically stores objects in files on disk
so that files are rewritten when objects are changed, and
so that objects are accessible but do not occupy memory
until they are accessed. Keeps track of times when objects
are created and modified, and caches some basic
characteristics of objects to allow for fast summaries of
objects. Also provides a command history mechanism that
saves the last command to a history file after each
command completes.

License GPL

Depends R (>= 2.1.0), methods

Suggests scriptests

NeedsCompilation no

Repository CRAN

Date/Publication 2016-07-23 00:05:49

R topics documented:

show.envs 2
track-intro 3
track.attach 10
track.auto 11
track.copy 12
track.design 14
track.future 19
track.history 22
track.info 24

track.manage 26
track.options 29
track.performance 33
track.plugin.lru 34
track.preremove 36
track.rebuild 37
track.rename 39
track.setup 40
track.status 44
track.summary 46
track.sync 49

Index **52**

`show.envs` *Show the environments referenced within an object.*

Description

`show.envs` recursively examines `x` and the objects within it, printing the names of any environments encountered. It does NOT recursively enter environments – once it finds an environment it just prints the name of that environment and doesn't look further inside the environment.

Usage

```
show.envs(x, obj = substitute(x))
```

Arguments

`x`	The object to examine.
`obj`	An expression describing the object. Not intended to be supplied by the user.

Details

`show.envs` attemps to show the environments referenced within an object, but it may miss some. If you encounter any such cases, please email them to <`tplate@acm.org`>.

Value

The number of environments encountered.

Author(s)

Tony Plate <`tplate@acm.org`>.

Examples

```
x <- data.frame(a=1:10, b=10:1)
m <- lm(a ~ b, data=x)
show.envs(m)
```

track-intro *Overview of track package*

Description

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.

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.

Details

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.

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. 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.

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

```
> library(track)
> # By default, track.start() uses/creates a db in the dir
> # 'rdatadir' in the current working directory; supply arg
> # dir= to change.
> track.start()
> x <- 123                  # Variable 'x' is now tracked
> y <- matrix(1:6, ncol=2)  # 'y' is assigned & tracked
> z1 <- list("a", "b", "c")
> z2 <- Sys.time()
> track.summary(size=F)     # See a summary of tracked vars
            class    mode extent length            modified TA TW
x         numeric numeric    [1]      1 2007-09-07 08:50:58  0  1
y          matrix numeric  [3x2]      6 2007-09-07 08:50:58  0  1
z1           list    list  [[3]]      3 2007-09-07 08:50:58  0  1
z2 POSIXt,POSIXct numeric    [1]      1 2007-09-07 08:50:58  0  1
> # (TA="total accesses", TW="total writes")
> ls(all=TRUE)
[1] "x"  "y"  "z1" "z2"
> track.stop(pos=1)              # Stop tracking
> ls(all=TRUE)
character(0)
>
> # Restart using the tracking dir -- the variables reappear
> track.start() # Start using the same tracking dir again ("rdatadir")
> ls(all=TRUE)
[1] "x"  "y"  "z1" "z2"
> track.summary(size=F)
            class    mode extent length            modified TA TW
x         numeric numeric    [1]      1 2007-09-07 08:50:58  0  1
y          matrix numeric  [3x2]      6 2007-09-07 08:50:58  0  1
z1           list    list  [[3]]      3 2007-09-07 08:50:58  0  1
z2 POSIXt,POSIXct numeric    [1]      1 2007-09-07 08:50:58  0  1
> track.stop(pos=1)
>
> # the files in the tracking directory:
> list.files("rdatadir", all=TRUE)
[1] "."                     ".."
[3] "filemap.txt"           ".trackingSummary.rda"
[5] "x.rda"                 "y.rda"
```

```
[7] "z1.rda"              "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).

List of basic functions and common calling patterns

For straightforward use of the track package, only a single call to `track.start()` need be made to start automatically tracking the global environment. If it is desired to save untrackable variables at the end of the session, `track.stop()` should be called before calling `save.image()` or `q('yes')`, because `track.stop()` will ensure that tracked variables are saved to disk and then remove them from the global environment, leaving `save.image()` to save only the untracked or untrackable variables. The basic functions used in automatic tracking are as follows:

- `track.start(dir=...)`: start tracking the global environment, with files saved in `dir` (the default is `rdatadir`).
- `track.summary()`: print a summary of the basic characteristics of tracked variables: name, class, extent, and creation, modification and access times.
- `track.info()`: print a summary of which tracking databases are currently active.
- `track.stop(pos=, all=)`: stop tracking. Any unsaved tracked variables are saved to disk. Unless `keepVars=TRUE` is supplied, all tracked variables become unavailable until tracking starts again.
- `track.attach(dir=..., pos=)`: attach an existing tracking database to the search list at the specified position. The default when attaching at positions other than 1 is to use readonly mode, but in non-readonly mode, changes to variables in the attached environment will be automatically saved to the database.
- `track.rescan(pos=)`: rescan a tracking directory that was attached by `track.attach()` at a position other than 1, and that is preferably readonly.

For the non-automatic mode, four other functions cover the majority of common usage:

- `track.start(dir=..., auto=TRUE/FALSE)`: start tracking the global environment, with files saved in `dir`
- `track(x)`: start tracking `x` - `x` in the global environment is replaced by an active binding and `x` is saved in its corresponding file in the tracking directory and, if caching is on, in the tracking environment

- `track(x <- value)`: start tracking `x`
- `track(list=c('x', 'y'))`: start tracking specified variables
- `track(all=TRUE)`: start tracking all untracked variables in the global environment
- `untrack(x)`: stop tracking variable `x` - the R object `x` is put back as an ordinary object in the global environment
- `untrack(all=TRUE)`: stop tracking all variables in the global environment (but tracking is still set up)
- `untrack(list=...)`: stop tracking specified variables
- `track.remove(x)`: completely remove all traces of `x` from the global environment, tracking environment and tracking directory. Note that if variable `x` in the global environment is tracked, `remove(x)` will make `x` an "orphaned" variable: `remove(x)` will just remove the active binding from the global environment, and leave `x` in the tracked environment and on file, and `x` will reappear after restarting tracking.

Complete list of functions and common calling patterns

The `track` package provides many additional functions for controlling how tracking is performed (e.g., whether or not tracked variables are cached in memory), examining the state of tracking (show which variables are tracked, untracked, orphaned, masked, etc.) and repairing tracking environments and databases that have become inconsistent or incomplete (this may result from resource limitiations, e.g., being unable to write a save file due to lack of disk space, or from manual tinkering, e.g., dropping a new save file into a tracking directory.)

The functions that can be used to set up and take down tracking are:

- `track.start(dir=...)`: start tracking, using the supplied directory
- `track.stop()`: stop tracking (any unsaved tracked variables are saved to disk and all tracked variables become unavailable until tracking starts again)
- `track.dir()`: return the path of the tracking directory

Functions for tracking and stopping tracking variables:

- `track(x)` `track(var <- value)` `track(list=...)` `track(all=TRUE)`: start tracking variable(s)
- `track.load(file=...): load some objects from    a RData file into the tracked environment`
- `untrack(x, keep.in.db=FALSE)` `untrack(list=...)` `untrack(all=TRUE)`: stop tracking variable(s) - value is left in place, and optionally, it is also left in the the database

Functions for getting status of tracking and summaries of variables:

- `track.summary()`: return a data frame containing a summary of the basic characteristics of tracked variables: name, class, extent, and creation, modification and access times.
- `track.status()`: return a data frame containing information about the tracking status of variables: whether they are saved to disk or not, etc.
- `track.info()`: return a data frame containing information about which tracking dbs are currently active.
- `track.mem()`: return a data frame containing information about number of objects and memory usage in tracking dbs.

- `env.is.tracked()`: tell whether an environment is currently tracked

The remaining functions allow the user to more closely manage variable tracking, but are less likely to be of use to new users.

Functions for getting status of tracking and summaries of variables:

- `tracked()`: return the names of tracked variables
- `untracked()`: return the names of untracked variables
- `untrackable()`: return the names of variables that cannot be tracked
- `track.unsaved()`: return the names of variables whose copy on file is out-of-date
- `track.orphaned()`: return the names of once-tracked variables that have lost their active binding (should not happen)
- `track.masked()`: return the names of once-tracked variables whose active binding has been overwritten by an ordinary variable (should not happen)

Functions for managing tracking and tracked variables:

- `track.options()`: examine and set options to control tracking
- `track.load()`: load variables from a saved RData file into the tracking session
- `track.copy()` and `track.move()`: copy or move variables from one tracking db to another
- `track.rename()` rename variables in a tracking db
- `track.rescan()`: reload variable values from disk (can forget all cached vars, remove no-longer existing tracked vars)
- `track.auto()`: turn auto-tracking on or off

Functions used internally as part of auto-tracking (generally not called by the user when auto-tracking is running):

- `track.sync()`: write unsaved variables to disk, and remove excess objects from memory. This function can be called by the user if they wish to remove excess objects from memory during a memory-intensive top-level command.
- `track.sync.callback()`: calls `track.sync()`, this function is installed as a task callback (to be called each time a top-level task is completed, see `taskCallback`). This function is not exported by the track package.
- `track.auto.monitor()`: an additional callback that monitors the existence of the callback to `track.sync.callback` and re-instates it if missing. This function is not exported by the track package.

Lower-level functions for managing tracking and tracked variables (generally not called by the user when auto-tracking is running):

- `track.remove()`: completely remove all traces of a tracked variable
- `track.save()`: write unsaved variables to disk
- `track.flush()`: write unsaved variables to disk, and remove from memory
- `track.forget()`: delete cached versions without saving to file (the object saved in the file will be retrieved next time the variable is accessed)

Functions for recovering from errors (caused by bugs or by multiple sessions updating bookkeeping data):

- `track.rebuild()`: rebuild tracking information from objects in memory or on disk

Design and internals of tracking:

- See help page `track.design`

Note

Some special kinds of objects don't work properly if referenced as active bindings and/or stored in a save file. One example is RODBC connections. To make it easy to work with such objects, two ways of excluding variables from automatic tracking are provided: the `autoTrackExcludePattern` option (a vector regular expressions: variables whose name match one of these will not be tracked); and the `autoTrackExcludeClass` option (a vector of class names: variables whose class matches one of these will not be tracked). New values can be added to these options as follows:

```
track.options(autoTrackExcludePattern="regexp")
track.options(autoTrackExcludeClass="classname")
```

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.)

Author(s)

Tony Plate <tplate@acm.org>

References

Roger D. Peng. Interacting with data using the filehash package. R News, 6(4):19-24, October 2006. `http://cran.r-project.org/doc/Rnews`

David E. Brahm. Delayed data packages. R News, 2(3):11-12, December 2002. `http://cran.r-project.org/doc/Rnews`

See Also

Design of the `track` package.

Potential future features of the `track` package.

Documentation for `save` and `load` (in 'base' package).

Documentation for `makeActiveBinding` and related functions (in 'base' package).

Inspriation from the packages `g.data` and `filehash`.

Description of the facility (`addTaskCallback`) for adding a callback function that is called at the end of each top-level task (each time R returns to the prompt after completing a command): `http://developer.r-project.org/TaskHandlers.pdf`.

Examples

```
##############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed from
# the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
##############################################################

library(track)
# Start tracking the global environment using a tmp directory
# Default tracking db dir is 'rdatadir' in the current working
# directory; omit the dir= argument to use this.
if (!is.element('tmpenv', search())) attach(new.env(), name='tmpenv', pos=2)
assign('tmpdatadir', pos='tmpenv', value=file.path(tempdir(), 'rdatadir1'))
track.start(dir=tmpdatadir)
a <- 1
b <- 2
ls()
track.status()
track.summary()
track.info()
track.stop()
# Variables are now gone because default action of track.stop()
# is to not read all tracked variables into memory (this could
# exhaust memory and/or be very time consuming).
ls()
# bring them back
track.start(dir=tmpdatadir)
ls()
# It is possible to keep tracked vars after stopping tracking:
track.stop(keepVars=TRUE)
ls()
```

`track.attach` *Attach a tracking database to the search path.*

Description

Attach a tracking database to the search path at a position other than 1. Variables in the tracking database are made available through a new environment attached at `pos` on the search path.

Usage

```
track.attach(dir, pos = 2, name = NULL,
             create = FALSE, readonly = !create,
             lockEnv = FALSE, verbose = TRUE, auto = NULL,
             dup.ok = FALSE)
track.detach(pos = NULL, name = NULL, detach = TRUE)
```

Arguments

`dir`	The directory where the tracking database resides.
`pos`	The position on the search path to attach the new environment at, or where it is currently attached.
`name`	The name to use on the search path for the new environment.
`create`	Should the tracking database be created if it does not exist?
`readonly`	Logical flag indicating whether the tracking db should be attached in a readonly mode. The global environment (`pos=1` in the search path) cannot be tracked in a readonly mode.
`lockEnv`	Should the environment be locked for a readonly tracking environment? The default is `FALSE` because locking the environment is irreversible, and it prevents rescanning or caching (because can't delete or add bindings)
`verbose`	print a message about what directory is being tracked?
`detach`	If `TRUE`, the environment attached to the search path (in a position other than 2) will be detached after stopping tracking, IF it was created by `track.attach()` and if there are no variables left remaining in the environment after removing all tracked variables. If `detach="force"`, the attached environment will be removed even if there are variables remaining in it (though not if it was not created by `track.attach`).
`auto`	Should auto-tracking be used? (see `track.start`).
`dup.ok`	Is it OK to proceed if the tracking database is already attached on the search path? If `dup.ok=FALSE` (the default) and the tracking database is already attached, the function issues a warning and does nothing.

Details

`track.attach` attaches a new environment to the search path at the specified position and variables in the tracking database are made available in it through lazy loading. Using `readonly==TRUE` ensure that no changes at all are made to the tracking database, and the environment is locked to ensure that variables cannot be created or deleted.

`track.detach` syncs all variables to files and detaches the environment from the search list if it is empty. See `track.stop` for conditions under which the environment may not be detached.

Value

NULL

Author(s)

Tony Plate <`tplate@acm.org`>

See Also

`track.start`

Examples

```
## Not run:
track.attach("path/to/tracking-database", pos=2, name="trackdb")
track.detach(pos=2)

## End(Not run)
```

`track.auto` *Query or set the status of automated tracking*

Description

Query or set the status of automated tracking

Usage

```
track.auto(auto = NULL, pos = 1, envir = as.environment(pos))
```

Arguments

`auto`	`NULL` the default (to query the status of automatic tracking), or logical (`TRUE` or `FALSE`) to set it.
`pos`	The search path position of the environment being tracked (default is 1 for the global environment)
`envir`	The environment being tracked. This is an alternate way (to the use of pos=) of specifying the environment being tracked, but should be rarely needed.

Value

A logical value, indicating whether automatic tracking is on or off.

Author(s)

Tony Plate <tplate@acm.org>

See Also

track.start

Examples

```
## Not run:
track.auto() # query the status of auto-tracking
track.auto(FALSE) # turn auto-tracking off
track.auto(TRUE) # turn auto-tracking on

## End(Not run)
```

---

track.copy *Copy or move objects from one tracking db to another*

Description

Copy or move objects from one tracking db to another.

Usage

```
track.copy(from, to = 1, list = NULL, pattern = NULL, glob = NULL, delete = FALSE,
         clobber = FALSE, skipExisting = FALSE, verbose = TRUE, do.untrackable = FALSE)
track.move(from, to = 1, list = NULL, pattern = NULL, glob = NULL, delete = TRUE,
         clobber = FALSE, skipExisting = FALSE, verbose = TRUE, do.untrackable = FALSE)
```

Arguments

`from`	The position on the search list of the source db. Can be numeric or the name as returned by `search()`. Can be a vector, in which case the action is repeated for each element of `from`.
`to`	The position on the search list of the destination db. Can be numeric or the name as returned by `search()`.
`list`	A character vector of the objects to copy or move
`pattern`	A regular expression specifying the objects to copy or move
`glob`	A glob specifying the objects to copy or move
`delete`	Should the objects in the source db be removed? The default is `FALSE` for `track.copy` and `TRUE` for `track.move`.

`clobber`	Should the objects in the destination db be overwritten?
`skipExisting`	If `TRUE`, skip objects where an object of the same name already exists in the destination db .
`verbose`	If `TRUE`, write out what is being done.
`do.untrackable`	If `TRUE`, also copy untrackable objects. Not yet implemented.

Details

`track.copy` copies objects by copying their underlying files from the source tracking db to the destination tracking db (behavior for untracked or untrackable objects is different – see below). It also copies the object summary (preserving modification times etc.) Objects are not loaded in R when copied, and if a copied object is cached in the source or destination db, the cached copy will be removed.

The readonly status of both tracking dbs will be respected – the function will stop with an error before doing anything if a readonly flag conflicts with what it needs to do.

The metadata in source and destination tracking dbs (i.e., the file map and the object summary) are updated after each object is copied. Thus, if an error occurs during copying, e.g., due to filesystem permissions or lack of space, the source and destination dbs will be left in a consistent state.

`track.move` calls `track.copy` with a default of `delete=TRUE` to remove the source object.

NB: `do.untrackable` **is not yet implemented: untracked/untrackable objects are ignored.** By default, untrackable objects in the source tracking db will not be copied or moved, and a warning will be given. The rules for untrackability are those of the destination db, as specified by the tracking options `autoTrackExcludePattern` and `autoTrackExcludeClass` in that db. If `do.untrackable=TRUE`, untrackable objects will be copied or moved (with `delete` still controlling whether the original is left or removed.) Untrackable objects that are copied are left as an ordinary untracked object in the destination db.

Value

A character vector of the objects copied or moved.

Author(s)

Tony Plate <tplate@acm.org>

Examples

```
##############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
##############################################################

library(track)
```

```
# Track two environments and transfer objects from one to the other.
# Use tmp dirs for the tracking dbs.
track.start(dir=file.path(tempdir(), 'rdatadir2'))
track.attach(dir=file.path(tempdir(), 'rdatadir3'), pos=2, create=TRUE)
assign("x1", 1, pos=2)
assign("x2", 2, pos=2)
assign("y3", 3, pos=2)
assign("y4", 4, pos=2)
assign("z5", 5)
track.status(1)
track.status(2)
track.copy(from=2, pat="^x", clobber=TRUE)
ls(1)
ls(2)
track.move(from=2, pat="^y", clobber=TRUE)
track.move(from=1, to=2, pat="^z", clobber=TRUE)
ls(1)
ls(2)
c(x1, x2, y3, y4)
track.move(from=2, pat="^x", clobber=TRUE)
ls(1)
ls(2)
c(x1, x2, y3, y4)
track.status(1)
track.status(2)
track.detach(2)
# Would normally not call track.stop(), but do so here to clean up after
# running this example.
track.stop(keepVars=TRUE)
```

`track.design` *Design of a tracking environment*

Description

This document describes the layout of a tracking environment. Object tracking works by replacing a variable with an active binding, and keeping the actual value of the variable on disk and/or in another environment. Tracked objects are automatically resaved to disk when they are changed. Basic characteristics, such as class, size, extent, and creation and modification times are recorded in a summary of all tracked objects.

Details

Object tracking works by replacing a variable with an active binding, and keeping the actual value of the variable on disk and/or in another environment. Whenever the variable is fetched or assigned, the active binding is called, and it writes the object to disk if necessary, and records basic characteristics of the objects in a summary of all objects, including creation, modification and access times.

A tracking environment can be linked to one environment on the search path, but the tracking environment is not on the search path itself. An environment on the search path can only have one tracking environment linked to it. In standard use, variables are tracked automatically by a task callback function. Alternatively, variables to track can be registered with the tracking environment using the function `track()`.

Any user-created environment on the search path, or the global environment, can be tracked.

The format used to store R objects in files is the one used by `save()`/`load()` – the objects in those files can be read using `load()` if desired.

The various variables and files involved in tracking are as follows (assuming the RData suffix being used is "rda"). Note that the default tracked visible environment is the global environment.

```
Tracked Visible Environment
(on search list)          Tracking Environment
                          (not on search list)
+-----------------+   +-------------------------+
|     .trackingEnv|-->|           .trackingDir |---> Tracking Directory (files)
|                 |   |                         |         |
|          x (*)  |   |                   x (@) |         +- x.rda
|        abc (*)  |   |                 abc (@) |         +- abc.rda
|          Y (*)  |   |                   Y (@) |         +- _1.rda
|         x1      |   |                         |         |
|         x2      |   |                         |         |
|                 |   |        .trackingFileMap |         +- filemap.txt
|                |   |       .trackingSummary |        +- .trackingSummary.rda
|                 |   |        .trackingUnsaved |
|                 |   | .trackingSummaryChanged |
|                 |   |        .trackingOptions |
+-----------------+   +-------------------------+
```

- variables marked (*) are tracked and are actually an active binding that refers to the corresponding variable in the tracking environment. There can also be untracked variables in the visible tracked environment, but in the standard mode of operation these are detected by the end-of-task callback function and are immediately converted to tracked variables (except for variables with reserved names like .trackingSummary, and variables matching exclude patterns, see options `autoTrackExcludePattern` and `autoTrackExcludeClass` in track.options.
- variables marked (@) may or may not exist – if they do not exist in the tracking environment, they will be automatically read from file when the corresponding tracked object is accessed.
- The "trackingEnv" attribute on the tracked environment is the tracking environment. This is implemented as an attribute on the tracked environment rather than as a variable in the tracked environment so that `save.image()` on the tracked environment will ignore the tracking environment. If the tracking environment were stored as a variable in the tracked environment, `save.image()` could end up storing two copies of every tracked variable: one when it accessed the active binding (it stores a copy of the object: `save()` doesn't know it's an active binding); and another if the object is cached in the tracking environment.
- The "trackingDir" attribute on the tracking environment specifies the absolute pathname of the directory under which tracked objects are stored on file. It uses the absolute pathname because the current directory of the R session can be changed using `setwd()`, which would result in losing a relative pathname.

- `.trackingFileMap` stores the base part of the file name corresponding to each tracked object as a named character vector (the names on the vector are the object names). Objects that do not have simple names have an associated file name like "_NNN" where "NNN" is a number. For example, the `.trackingFileMap` for the above configuration could be `c(abc="abc", x="x", Y="_1")`. Simple object names are those conforming to the following rules:
  - less than 55 characters
  - are comprised of only lower-case letters, digits 0 through 9, "." and "_"
  - begin with a lower-case letter
  - are not one of the following: `con`, `prn`, `aux`, `nul`, `com1` through `com9`, `lpt1` through `lpt9`, and do not begin with one of these names followed by a period (i.e., `prn.foo` and `prn.foo.bar` are both not simple names) (these are special file names under Microsoft Windows - see http://en.wikipedia.org/wiki/Filename and search the web on the keywords "windows short file names rules prn com" to find an official Microsoft site.)

  The object `.trackingFileMap` is always kept in memory and is always saved to disk (as text in the file `filemap.txt`) whenever it is changed.
- `.trackingSummary` is a data frame recording various basic characteristics of the tracked objects, such as class, size and extent, and also times of creation, and most recent modification and access. The tracking summary should be accessed using the function `track.summary()`.
- `.trackingSummaryChanged` A logical flag indicating whether or not the tracking summary copy on disk is in sync with version in memory. To reduce overhead on accessing objects, there is an option to not resave the tracking summary when it is changed on accessing an object – this variable indicates if it has been changed.
- `.trackingUnsaved`: If the tracking options are set up so that objects are not automatically written to files on assignment, this variable contains a vector of names of all objects that have not been saved.
- `.trackingOptions`: are accessed and changed by the `track.options()` function. They are kept in memory, and also written to disk whenever they are changed. The tracking directory is organized as an R package. It's layout is as follows (saying, for example, that `attr(trackingEnv, "trackingDir")` is `/tmp/trackdir1`, and `.trackingFileMap` is `c(abc="abc", x="x", Y="_1`

```
/tmp/trackdir1
      |
      +- filemap.txt
      +- .trackingSummary.rda
      +- x.rda
      +- abc.rda
      +- _1.rda
```

Terminology

One could describe a tracking environment as "attached" to the tracked environment, but that using that term would risk confusion with the role of the `attach()` function and search path in R. So, instead the `track` package says that a tracking environment is "linked" to the tracked environment.

track: The `track` *tracks* variables, by setting up a one-to-one relationship between R objects and files on disks so that when an object in R is modified, the file on disk is automatically updated.

tracked environment: A *tracked* environment contains user variables and is usually on the search path.

tracked object: A *tracked* object (in a tracked environment) that has an active binding so that when it is modified, the corresponding file on disk is also modified.

untracked object: An *untracked* object in a tracked environment is an ordinary object that is not tracked and has no corresponding file.

tracking environment: A *tracking* environment is a special environment used by the `track` package to track objects in the *tracked* environment

linked: A tracking environment is *linked* to a tracked environment (by the `trackingEnv` attribute on the tracked environment, which *points* to the tracking environment.)

start tracking, stop tracking: Tracking is *started* by creating a tracking environment, linking it to the tracked environment, and setting up bindings for tracked objects.

tracking database: A *tracking* database is the collection of files and directories that stores the tracking information.

active tracking database: A *tracking* database that is currently linked to an environment in a running R session.

Untrackable variables – reserved names

Only ordinary variables can be tracked – variables that are active bindings cannot be tracked.

Several variable names are reserved and cannot be tracked: `.trackingEnv`, `.trackingFileMap`, `.trackingUnsaved`, `.trackingSummary`, `.trackingSummaryChanged`, `.trackingOptions`. Additionally, any variable with a newline character ("\n") as part of its name cannot be tracked (the main reason for this is that the mapping from object names to file names is stored in a text file, and newline character delimits the name).

The file map

The mapping from object names to file names is stored in the file `fileMap.txt`. This data is stored as ordinary text file to make it easy for users to see the object-file mappings outside of R.

Implementation considerations

The reason that objects must be explicitly registered for tracking is that there is currently no way of setting up a function to be called when a new object is created, so new objects are always created as ordinary R objects. Similarly, the R `remove()` functions does not have any hooks, so if `remove()` is called on a tracked variable, it will just remove the active binding in the visible environment, but will not disturb the underlying tracking environment. The `track.remove()` function will completely remove a tracked variable from the visible environment and the underlying tracking environment (including deleting an associated disk file.)

Object tracking was intended to be used in situations where large numbers of large objects must be manipulated. Consequently, there is a good chance of exhausting resources while using the `track` package. The `track` code tries to check return codes when creating objects or writing files, and in cases where it is unable to complete an operation it tries leave the tracking environment in a state from which objects can be salvaged. The functions `track.rebuild()` and `track.flush()` are provided to help recover from situations where resource limitations prevented successful operation.

Note that files are generally written in a "unsafe" manner (i.e., existing files can be overwritten with partial new files), but in these cases data is retained in the memory and can be rewritten after resolving file system problems.

The R functions `exists()` should be used with care on tracked objects, because it will actually fetch the object, possibly needing to read it from disk. In the `track` code, the `exists("x")` function is not used to check existence of a possibly tracked object `x`, instead an idiom like `is.element("x", objects(all=TRUE))` is used.

These statements about the available facilities in R were true as of R-2.4.1 (released Dec 2006).

The rules for how variable names are mapped to file names are based on trying to use filenames that will work properly on all three operating systems R works on (Linux, Windows, and Mac OS X). A somewhat obscure point that must be taken into account is the case-insensitivity of Mac OS X and Windows. Even though modern versions of the OS's seem to use case in their file names, this is because they are case preserving, but they are in fact still case insensitive. This means that a file created with the name "X.rda" is the same file as the "x.rda". Here is a short shell transcript showing this behavior in a bash shell running under Windows and Mac OS X (it's the same in both).

```
$ echo 123 > X
$ cat x
123
$ echo 456 > x
$ cat x
456
$ cat X
456
```

Thus, in order to work on OS's, file mapping must be used to create different filenames for the R objects "x" and "X" (which are in fact different in R.)

Portability

Tracking directories are intended to be operating-system independent and completely portable across different operating systems.

Compression

Saved R objects are compressed by default in R and by the `track` package. Decompression speed is very important for interactive response when using track, because each time an object is accessed, it is read from its file (unless the object is cached). Of the compression algorithms available as of R-2.12.0, which are gzip, bzip2, and xz, gzip is the winner in terms of speed. The default compression level in R for gzip is 6, but level 1 gives faster compression with slightly larger files (though decompression is not faster). The lzop compression algorithm `http://www.lzop.org` is still faster but it is not yet available in R.

Here are some comparisons and benchmarks of various compression programs:

- `http://www.linuxjournal.com/node/8051/print`
- `http://tukaani.org/lzma/benchmarks.html`

- http://stephane.lesimple.fr/wiki/blog/lzop_vs_compress_vs_gzip_vs_bzip2_vs_lzma_vs_lzma2-xz_benchmark_reloaded
- http://aliver.wordpress.com/2010/06/22/huge-unix-file-compresser-shootout-with-tons-of-datagra
- http://www.maximumcompression.com/
- http://mattmahoney.net/dc/text.html

Compression/decompression is nicely handled in R: only the call to `save()` has arguments for compression. Decompression in `load()` is handled automatically using a standard code (magic) at the start of the saved file. Saved files can also be compressed or decompressed outside of R, and `load()` will still handle them correctly, provided the compression used is one of the types that R knows about.

Author(s)

Tony Plate <tplate@acm.org>

References

Roger D. Peng. Interacting with data using the filehash package. R News, 6(4):19-24, October 2006. http://cran.r-project.org/doc/Rnews.

David E. Brahm. Delayed data packages. R News, 2(3):11-12, December 2002. http://cran.r-project.org/doc/Rnews

See Also

Overview of the `track` package.

Documentation for `makeActiveBinding` and related functions (in 'base' package).

Inspriation from the packages `g.data` and `filehash`.

`track.future` *Potential future features of the track package*

Description

Potential future features of the track package, in some vague order of feasibility and priority ('easy', 'medium' and 'hard' are an estimate of design and coding difficulty):

better handling of writing files on changes to objects: (medium) with cachePolicy="tltPurge", changed objects are only written to file at the end of a top level task. However, with cachePolicy="none", objects are written to file on each change – is better control over this needed?

untracked variables in the summary: (easy) would this be useful? wouldn't need to cache these, mark with an asterisk in a special column? Compute these each time track.summary is called.

option writeToDisk: is this redundant option with cache and cachePolicy?

other default tracked environment: (easy) would it be useful to allow an environment other than the global environment to be the default tracking environment? This could be implemented by using `options("tracked.environment")` as the default environment for all the tracking functions (rather than the currently hardcoded pos=1)

DONE better cleanup: (easy) provide an integrated quiting function that saves all tracked vars and history before quitting (and maybe also saves untracked vars in an RData file)

caching rules: (hard) allow rule-based decisions for caching, e.g., only cache objects under a certain size, or only cache objects of certain classes, or enforce a limit on memory for caching tracked variables, and flush out least-recently used variables

record file read/writes: (easy) record each time a file is read or written in the summary. Could be useful for smarter caching.

auto-trust in rebuild: (easy) when rebuilding an active tracking environment, base decision whether to use summary row from file or environment on which has more recent dates in it. (whole dataframe, or row by row?)

smarter reading of filemap.txt: (medium) check the mod time on filemap.txt when getting the filemap obj, and if the file on disk appears to have changed, reread it instead of just getting it from memory. This would allow working together better with other sessions that are simultaneously using this tracking dir. Don't know how much it would slow things down – do some timings. Note that to make this work in a fool-proof manner would require locks.

investigate double-get: doing subset-replacement (e.g., `X[2] <- ...`) retrieves `X` twice (see example below)

DONE readonly mode: (hard) to allow linking tracking dirs that might be in use by other R processes – would require not recording gets – this would require adding a new env on the search path and tracking it

DONE autoflush: (hard) automatic flushing of variables that haven't been used frequently (triggered automaticall when memory runs low?) – this is why the summary records fetches as well as writes

safer restart: (medium) check that we will be able to restart before doing the stop (check for masked variables or other potential clobber problems)

safe saves: (hard) write files in a safe way so that the original file is not removed until the new file is written – not sure if this is necessary, because objects are in memory, and can be rewritten if there is a failure

DONE autotrack: (hard) automatically track new variables? (would require hooks in base-R that get called when a new var is created)

Example of the "double-get" when assigning a subset (using the example from the help page for `makeActiveBinding`). Note that it works correctly, but retrieving the object twice seems unneccessary and could be slow with very large objects.

```
> f <- local( {
+     x <- 1
+     function(v) {
+        if (missing(v))
+            cat("get\n")
+        else {
```

```
+             cat("set\n")
+             x <<- v
+         }
+         x
+     }
+ })
> makeActiveBinding("X", f, .GlobalEnv)
NULL
> bindingIsActive("X", .GlobalEnv)
[1] TRUE
> X
get
[1] 1
> X <- 2
set
> X
get
[1] 2
>
> X[1]
get
[1] 2
> X[2] <- 1 # 'X' is fetched twice
get
get
set
> X
get
[1] 2 1
>
```

See Also

Overview and design of the `track` package.

Examples

```
# Example (transcript shown above) of how subset-assignment
# results in two retrievals when the object is an active binding.
f <- local( {
    x <- 1
    function(v) {
       if (missing(v)) {
           cat("get\n")
       } else {
           cat("set\n")
           x <<- v
       }
       x
    }
})
makeActiveBinding("X", f, .GlobalEnv)
bindingIsActive("X", .GlobalEnv)
X
X <- 2
X
X[1]
X[2] <- 1 # 'X' is fetched twice
X
```

`track.history` *Functions for incrementally writing command history to a file.*

Description

These functions provide the ability to append recent commands to a history file after (almost) every top level command. This makes it unnecessary to use `savehistory()` and solves the problem of command history not being saved on accidental or abnormal termination of an R session.

Usage

```
track.history.start(file = NULL, width = NULL, style = NULL, times =
  NULL, load = TRUE, verbose = FALSE, message="Session start")
track.history.stop()
track.history.status()
track.history.load(times = FALSE)
track.history.writer(expr, value, ok, visible)
```

Arguments

`file`	File to store the incremental history in. Defaults to `.Rincr_history`.
`width`	Width to use deparsing for fast-style history saving. Defaults to `120`.
`style`	Two styles are possible: • `full`: (default) Use the internal R history mechanism. This is slower, but it records everything typed at the command prompt, in the original formatting. • `fast`: Use deparsed version of the most recently executed command. This is fast, but it doesn't record comments and some commands with errors, and it changes formatting.
`times`	Should time stamps be written to or read from the file? The default behavior for `track.history.start()` is `TRUE`. The default when loading history is `FALSE`, so that time stamps do not appear in the interactive history.
`load`	Should existing history be loaded when starting incremental history?
`verbose`	Should comments be printed?
`expr`	Provided by the task callback
`value`	Provided by the task callback

`ok`	Provided by the task callback
`visible`	Provided by the task callback
`message`	A string to be written (once) to the incremental history file along with the date and time when incremental history tracking is started.

Details

Default values are taken first from options `incr.hist.style`, `incr.hist.width`, `incr.hist.file` and `incr.hist.times`. If those option values don't exist, values are taken from environment variables `R_INCR_HIST_STYLE`, `R_INCR_HIST_WIDTH`, `R_INCR_HIST_FILE`, `R_INCR_HIST_TIMES`.

- `track.history.start()` installs `track.history.writer()` as a task callback handler.
- `track.history.load()` loads history from the file that incremental history is being written to.
- `track.history.stop()` removes the task callback handler.
- `track.history.writer()` is the task callback handler – it is not intended to be called by the user.

If arguments are supplied to `track.history.start()`, their values are remembered in `options()` and used for the remainder of the session or until changed.

The history stored using `style="full"` is more complete and accurate, in that it includes comments, unparseable commands, and original formatting. It is somewhat slower because it is based on the internal history mechanism, which doesn't provide an easy way of identifying which are the new commands. Consequently, when using `style="full"` `track.history.writer()` must inspect the entire internal history at the end of each command to work out which lines in it have been added since the last time history was written. However, the time difference seems negligible for interactive use on ordinary workstations circa 2010.

To set up incremental history tracking automatically, put the following in your `.Rprofile`:

```
if (interactive()) {
    track.history.load()
    track.history.start()
}
```

Value

`track.history.status()` returns a character string: `"on"` or `"off"`. The other functions currently provide no useful return values.

Author(s)

Tony Plate <`tplate@acm.org`>

See Also

`addTaskCallback` To read in command history that is stored in a particular file, use `loadhistory(file)`. `savehistory`

Examples

```
## Not run:
## Can't use history except in Rgui and Rterm
track.history.start()

## End(Not run)
```

`track.info` *Return filenames and directories for tracked variables.*

Description

Return filenames and directories for tracked variables.

Usage

```
track.filename(expr, list = character(0), pos = 1,
               envir = as.environment(pos), suffix = FALSE)
track.datadir(pos = 1, envir = as.environment(pos), relative = TRUE)
track.info(pos = NULL, envir = as.environment(pos), all=is.null(pos))
track.mem(pos = NULL, envir = as.environment(pos), all=is.null(pos))
env.is.tracked(pos = 1, envir = as.environment(pos))
tracked.envs(envirs=search())
```

Arguments

`expr`	An unquoted variable name
`list`	A character vector of variable names
`pos`	The search path position of the environment being tracked (default is 1 for the global environment)
`envir`	The environment being tracked. This is an alternate way (to the use of `pos=`) of specifying the environment being tracked, but should be rarely needed.
`suffix`	: Return the filename with the RData suffix (extension) (taken from `track.options("RDataSuffix")`)
`relative`	: Return a path relative to the current working directory, or an absolute path?
`all`	Return info about all tracked environments?
`envirs`	A list or vector of objects that can be interpreted as environments by `as.environment`

Value

track.filename() returns the filenames for tracked variables. These names are guaranteed to be distinct for distinct variables.

track.datadir() returns the directory in which RData files for tracked variables are stored.

`track.info`**:** returns a dataframe of information (directory, readonly/writable status) about environments currently tracked.

`track.mem`**:** returns a dataframe of information (number of objects, memory usage) about environments currently tracked.

env.is.tracked: returns `TRUE` or `FALSE`

tracked.envs: with no arguments, it returns the names of tracked environment that are on the search list. If given an argument that is a vector of environments (or environment names), it returns the subset of that vector that are tracked environments.

Note

The `track` package stores RData files in the directory returned by `track.datadir()`. It is not advisable to write other RData files to that directory. Filenames for variables may change when an object is deleted and then recreated.

A warning message like "env R_GlobalEnv (pos 1 on search list) appears to be an inactive tracked environment, saved from another session and loaded here inappropriately" indicates that the environment has some but not all of the structure of a tracked environment. In particular, the variable `.trackingEnv` exists in it, but does not seem to be connected properly. Some of the bindings may be active bindings, but they may have come disconnected from the tracking environment. The most common way that this kind of situation can arise is from doing `save.image()` before `track.stop()`, and then reloading the saved image (e.g., when restarting R). To fix this situation, do the following:

1. `rm(.trackingEnv, pos=1)`
2. `names(which(!sapply(ls(pos=1), bindingIsActive, as.environment(1))))` # to see which variables have active bindings
3. `x1 <- x` # for each variable x that has an active binding and that you want to save
4. `rm(x, pos=1)`
5. `save.image()` # to overwrite the old saved .RData file (only works with position 1)

If the inactive tracked environment is at a position other than 1 on the search list, substitute the appropriate position for 1 in the above.

Author(s)

Tony Plate <`tplate@acm.org`>

See Also

Overview and design of the `track` package.

Examples

```
###############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
```

```
##################################################################

library(track)
track.start(dir=file.path(tempdir(), 'rdatadir4'))
x <- 33
X <- array(1:24, dim=2:4)
Y <- list(a=1:3,b=2)
X[2] <- -1
track.datadir(relative=TRUE)
track.datadir(relative=FALSE)
track.filename(list=c("x", "X"))
track.info()
track.mem()
env.is.tracked(pos=1)
env.is.tracked(pos=2)
# Would normally not call track.stop(), but do so here to clean up after
# running this example.
track.stop(pos=1, keepVars=TRUE)
```

track.manage *Manage how objects are handled in a tracking session*

Description

Functions to start and stop tracking objects, remove them, load objects from RData files, and manage cached and saved copies of objects. These functions should not be needed in plain vanilla use of the track package.

For an introduction to the track package, see Overview (?track.intro).

Usage

```
track(expr, pos = 1, envir = as.environment(pos), list = NULL,
        pattern = NULL, glob = NULL, exclude = TRUE)
track.assign(x, value, pos = 1, envir = as.environment(pos), flush = TRUE)
untrack(expr, pos = 1, envir = as.environment(pos), list = NULL,
        pattern = NULL, glob = NULL, all = FALSE, keep.in.db = FALSE)
track.remove(expr, pos = 1, envir = as.environment(pos), list = NULL,
        pattern = NULL, glob = NULL, all = FALSE, force = TRUE)
track.save(expr, pos = 1, envir = as.environment(pos), list = NULL,
        pattern = NULL, glob = NULL,
       all = missing(expr) && missing(list) && missing(pattern) && missing(glob))
track.resave(expr, pos = 1, envir = as.environment(pos), list = NULL,
        pattern = NULL, glob = NULL,
       all = missing(expr) && missing(list) && missing(pattern) && missing(glob))
track.flush(expr, pos = 1, envir = as.environment(pos), list = NULL,
        pattern = NULL, glob = NULL,
       all = missing(expr) && missing(list) && missing(pattern) && missing(glob),
        force = FALSE)
```

```
track.forget(expr, pos = 1, envir = as.environment(pos), list = NULL,
        pattern = NULL, glob = NULL, all = FALSE)
track.load(files, pos = 1, envir = as.environment(pos), list = NULL,
        pattern = NULL, glob = NULL, cache = FALSE, clobber = FALSE,
        time.of.file = TRUE, warn = TRUE)
```

Arguments

`expr`	An unquoted variable name
`pos`	The search path position of the environment being tracked (default is 1 for the global environment)
`envir`	The environment being tracked. This is an alternate way (to the use of pos=) of specifying the environment being tracked, but should be rarely needed.
`list`	A character vector of variable names to operate upon
`pattern`	A regular expression specifying variable names to operate upon
`glob`	A regular expression specifying variable names to operate upon
`all`	If `TRUE`, operate upon all elegible variables. The default is `FALSE` for functions that can change data, and `TRUE` for functions that merely control whether data is in memory or file or both.
`exclude`	Controls exclusion of particular variables by pattern matching against a vector of regular expressions in the `autoTrackExcludePattern` option value. If `exclude==TRUE` (the default), exclude variables that match. If `exclude==FALSE`, ignore the exclusion patterns.
`keep.in.db`	If `TRUE`, the variable is left in the tracking database, though the link to it is broken (it becomes masked)
`files`	A vector of names of RData files (any file saved by `save()`)
`cache`	`TRUE` or `FALSE` indicating whether to keep the tracked object cached in memory
`clobber`	`TRUE` or `FALSE` indicating whether to overwrite existing objects of the same name
`force`	If `TRUE`, for `track.remove` remove orphaned tracked variables; for `track.flush` flush out variables that would normally be kept in cache.
`time.of.file`	If `TRUE`, use the access times on the file to populate the access time fields in the tracking summary.
`warn`	If `TRUE`, issue warnings about object not acted upon.
`x`	A variable name, as a character vector of length 1
`value`	The value to assign
`flush`	Logical value, specifying whether to flush the assigned object out of memory

Details

These functions are executed for their side effects:

- `track`: start tracking the specified variables
- `track.assign`: assign a value to a variable (start tracking variable if it is not already tracked.) Optionally flush the value out of memory.

- `untrack`: stop tracking the specified variables, leaving the object in `envir` so that it can still be used. If `keep.in.db=TRUE`, the variable is left in the tracking environment (but is masked), if `keep.in.db=FALSE` (the default), all trace of the variable is completely removed from the tracking environment.
- `track.remove`: completely remove all traces of a tracked variable (also removes untracked variables)
- `track.save`: write unsaved variables to disk
- `track.flush`: write unsaved variables to disk, and remove from memory
- `track.forget`: delete cached versions without saving to file (file version will be retrieved next time the variable is accessed)
- `track.rescan`: reload variable values from disk (can forget all cached vars, remove no-longer existing tracked vars)
- `track.load`: load variables from a saved RData file into the tracking session - if `list` is supplied, only these variables are loaded in. Already existing variables will be skipped and not overwritten unless `clobber=TRUE` is supplied.

The variables to be acted upon are specified either in `expr` (a variable name, unquoted) or `list` (character vector containing names of variables), or by regular expression `pattern` or shell pattern `glob`. If no specification is given, all variables are acted upon.

Value

The value returned from these functions is invisible and typically contains the names of objects acted upon.

`track:` a character vector containing the names of objects added to the tracking environment

`untrack, track.remove, track.save, track.flush, track.forget, track.rescan:`
a character vector containing the names of objects in the tracking environment that were acted upon

`track.load:` a list with two components:

- `loaded`: names of objects that were loaded from `file`
- `skipped`: names of objects in `file` that were not loaded

Author(s)

Tony Plate <tplate@acm.org>

See Also

Overview and design of the `track` package.

Examples

```
##############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
##############################################################

library(track)
track.start(dir=file.path(tempdir(), 'rdatadir5'))
x <- 33
X <- array(1:24, dim=2:4)
Y <- list(a=1:3,b=2)
X[2] <- -1
track.summary(time=0, access=1, size=FALSE)
y1 <- 2
y2 <- 3
z1 <- 4
z2 <- 5
z3 <- 6
untracked()
track.summary(time=0, access=1, size=FALSE)
ls(all=TRUE)
track.stop(pos=1)
ls(all=TRUE)
a <- 7
b <- 8
save(list=c("a", "b"), file=file.path(tempdir(), "ab.rda"))
remove(list=c("a", "b"))
track.start(dir=file.path(tempdir(), 'rdatadir5'))
track.summary(time=0, access=1, size=FALSE)
track.load(file.path(tempdir(), "ab.rda"))
track.summary(time=0, access=1, size=FALSE)
track.status()
# Would normally not call track.stop(), but do so here to clean up after
# running this example.
track.stop(pos=1, keepVars=TRUE)
```

`track.options` *Set and get tracking options on a tracked environment*

Description

Set and get tracking options on a tracked environment. Each tracked environment has its own set of tracking options exists which can be changed indpendently. Global default values can be set in `options("global.track.options")`.

Usage

```
track.options(..., pos = 1, envir = as.environment(pos),
              values=list(...), save = FALSE, clear=FALSE, delete=FALSE,
              trackingEnv, only.preprocess = FALSE, old.options = list())
```

Arguments

`...`	Either option names as character data, or specifications for setting options as named arguments or in a named list. See DETAILS for descriptions of options.
`pos`	The search path position of the environment being tracked (default is 1 for the global environment)
`envir`	The environment being tracked. This is an alternate way (to the use of `pos=`) of specifying the environment being tracked, but should be rarely needed.
`values`	A named list of option values to set. `track.options(readonly=T)` is equivalent to `track.options(values=list(readonly=TRUE))`
`save`	If `TRUE`, current options are saved to disk and will be used in future. Note that all current options settings are saved, not just the new settings made in this call.
`clear`	If `TRUE`, and the option can have multiple values (e.g., `autoTrackExcludeClass`), the current values are cleared prior to using the supplied values. The default behavior, with `clear=FALSE` and `delete=FALSE` is to add supplied values to multi-valued options, and to replace the value for single-valued options.
`delete`	If `TRUE`, and the option can have multiple values, the supplied values are removed from the current values (if they are not in the current values, they are silently ignored.)
`trackingEnv`	The hidden environment in which tracked objects are stored. It is not necessary to supply this in normal use.
`only.preprocess`	If `TRUE`, process any options specifications and return the full list of option settings with the values as specified, and defaults for all othe roptions. Stored options are neither accessed nor changed. Intended for internal use.
`old.options`	A list of old options to use, can only be suppled when `only.preprocess=TRUE`. Intended for internal use.

Details

Valid option names and values are as follows:

alwaysCache: character (default `".Last"`): vector of objects to always keep in memory. `".Last"` is here to avoid difficulties quitting R if the tracking DB becomes unavailable.

alwaysCacheClass: character (default `"ff"`): vector of classes whose objects are always kept in memory. `"ff"` is here by default because `"ff"` objects generally occupy only a small amount of memory, and flushing the object from memory causes unnecessary finalization calls on the external pointers in `"ff"` objects, which changes their behavior.

alwaysSaveSummary: logical (default TRUE) if TRUE, always save the summary on any change to the summary. Summaries are not saved for databases attached in a readonly mode.

autoTrackExcludeClass: character vector. Variables whose class is in this vector are not auto-tracked. The default is `"RODBC"`, because variables of that class do not work after being saved and reloaded.

autoTrackExcludePattern: character vector (default `c("^\.track", "^\.required")`) variables whose name matches any of these regular expressions are not auto-tracked

autoTrackFullSyncWait: (default -1) auto track will wait at least this many seconds between doing a full sync at the end of a top level task. If equal to zero, do a full sync at the end of each top level task. If less than zero, don't do a full sync. Doing a full sync can be slow, so this is off by default.

cache: logical (default TRUE): keep objects in memory?

cacheKeepFun: A function that specifies which objects to keep in memory at the end of a top-level-task. track.plugins for further info. Can be `"none"` or `NULL`.

cachePolicy: The higher-level policy to follow regarding keeping objects in memory. Currently has two possible values - one of them allows special action at the end of a top-level-task:

`"none"`**:** No special action at end of task, i.e., follow option `cache`

`"eotPurge"`**:** Purge objects from memory at the end of a top-level task

Also affects when changes to objects are written to disk - see option `writeToDisk` below.

clobberVars: vector of string specifying variables to be clobbered silently when attaching a tracking db

compress: character or logical (default `TRUE`) passed to `save()`. Possible values are `"none"`, `"gzip"`, `"xz"`, `"bzip2"`. `save()` currently uses gzip by default (i.e., when `compress=TRUE`), which according to `save()` offers the best tradeoff of filesize and compression and decompression times.

compression_level: numeric (default `1`) passed to `save()`

debug: integer (default 0) if > 0, print some diagnostic debugging messages

maintainSummary: logical (default TRUE) if TRUE, record time & number of changes and accesses

RDataSuffix: character (default `"rda"`) suffix to use for files containing saved R objects

readonly: logical (default TRUE for `track.attach()` and FALSE for `track.start()`) should any changes be allowed to the files? Note that this option is a function of how a tracking database is accessed – it is not a property of the database itself. A particular tracking database can attached on one R session with `readonly=TRUE` and at the same time be attached to another R session with `readonly=FALSE`. To unconditionally protect a tracking database from modification, use file permissions.

recordAccesses: logical (default TRUE) if TRUE, record counts and times for access ("get") operations on tracked variables

summaryAccess: logical, or integer value 0,1,2,3,4; controls what info about accesses is output by `track.summary()`

summaryTimes: logical, or integer value 0,1,2,3 (see `track.summary()` for the effect of these settings)

writeToDisk: logical (default TRUE): always write changed objects to disk? If `TRUE`, when objects are written to disk depends on `cachePolicy`: `cachePolicy="none"`: write objects immediately on a change; `cachePolicy="eotPurge"`: write changed objects at the end of a top-level task

The option settings are saved as a list in an object called `.trackingOptions` in the tracking environment (with a copy mirrored to a file in the tracking dir if `save=TRUE`.)

The options can be used to tune performance to resource availability (time & memory) and robustness in the face of machine or user error. Some possible settings are:

maximize robustness and speed: `cache=TRUE` and `writeToDisk=TRUE` (the default): always write an object to disk when it is changed, and keep a copy in memory, so that an object only needs to be read once

minimize memory usage and maximize robustness: `writeToDisk=TRUE`, `cache=FALSE`: always write an object to disk when it is changed, and don't keep a copy in memory – need to read from disk whenever the object is referred to

maximize speed: `writeToDisk=FALSE`, `cache=TRUE`: don't write the object to disk - just keep a copy in memory after it is first accessed and only write it when `track.stop()` or one of `track.save()` or its friends is called. This combination less robust because changed variables can be lost if R crashes, or the user quits R without remembering to call `track.stop()`. This mode of operation is like the `g.data` package, but with automatically keeping track of which variables have been changed and need to be written to disk (and the writing of changed variables with one call to `track.save()` or `track.stop()`).

The combination `writeToDisk=FALSE` and `cache=FALSE` is possible, but is unlikely to be desirable – this will keep changed objects in memory, but will not keep merely fetched objects in memory.

The options `maintainSummary`, `recordAccesses`, and `alwaysSaveSummary` control when the object summary is updated and when it is saved to disk (the default is for it to be updated and saved to disk for every read and write access to an object, whether or not the object is cached in memory).

Global default values can be set in `options("global.track.options")` as a list like `options(global.track.options=li`

Value

The value returned is a list of option values. If options were specified as arguments, the old values of those options are returned (unless `only.preprocess=TRUE` was supplied). If no options were specified as arguments, the full list of current option values is returned.

Cache plugin functions

`track` allows users to supply their own plugin functions that specify cache rules. The plugin function is called at the end of a top-level command. The default plugin function implements a rule that flushes least-recently accessed large objects from the cache when more memory usage is over a threshold. See track.plugins for further info.

Author(s)

Tony Plate <tplate@acm.org>

See Also

Overview and design of the `track` package. See track.plugins for description of cache plugin functions

Examples

```
##############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
##############################################################

library(track)
track.start(dir=file.path(tempdir(), 'rdatadir6'))
x <- 33
X <- array(1:24, dim=2:4)
track.status()
track.options(cache=TRUE, writeToDisk=FALSE) # change for just this session
# different ways of retrieving option values
track.options(c("cache", "writeToDisk"))
track.options("cache", "writeToDisk")
track.options("cache")
track.options()
# see the effect of the changed options on the status of X (X is not saved to disk)
track.status()
X[1,1,1] <- 0
track.status()
track.flush()
track.status()
track.stop(pos=1)
track.start(dir=file.path(tempdir(), 'rdatadir6'))
# note that options previously changed are back at defaults (because default
# to track.options() is save=FALSE
track.options(c("cache", "writeToDisk"))
track.options(cache=TRUE, writeToDisk=FALSE, save=TRUE) # change the options on disk
track.options(c("cache", "writeToDisk"))
track.stop(pos=1)
track.start(dir=file.path(tempdir(), 'rdatadir6'))
# now options previously changed are remembered (because track.options(..., save=TRUE) was used)
track.options(c("cache", "writeToDisk"))
track.stop(pos=1, keepVars=TRUE)
```

`track.performance` *Performance tuning with track.*

Description

Performance tuning with track involves trading off memory use for faster access times to objects. Access time is fastest when all objects are cached in memory, but memory can be exhausted if this is done. Memory use is minimized when objects are not cached in memory at all, but then a file must be read or written each time an object is referenced, and the whole file must be read or written even if only a small part of the object is actually used or changed.

The default mode of operation of track balances memory use and access times by keeping objects in memory for the duration of a top-level task, and flushing them out at the end of the task.

Three options (see track.options) control performance:

- cache: `TRUE`/`FALSE` should variables be cached at all?
- cachePolicy: `tltPurge`/`none` higher level policy for maintaining cache
- writeToDisk: `TRUE`/`FALSE` should objects be written after a change?

Useful possible combinations of settings are:

- cache=TRUE, cachePolicy="tltPurge", writeToDisk=TRUE: (DEFAULT) keep objects in memory for the duration of a task; flush and/or write to disk at the end of task
- cache=TRUE, cachePolicy="none", writeToDisk=TRUE: keep all objects in memory until removed with `track.flush`; write changed objects to disk immediately
- cache=TRUE, cachePolicy="none", writeToDisk=FALSE: keep all objects in memory until removed with `track.flush`; don't automatically write changed objects to disk (use `track.save`)
- cache=FALSE, cachePolicy="none", writeToDisk=TRUE: keep no objects in memory; write changed objects to disk immediately

Performance tuning is a possible area of future development of the track package, at as version 0.9-9, the defaults settings of `cache=TRUE`, `cachePolicy="tltPurge"`, and `writeToDisk=TRUE` work well. However, smarter caching based on access patterns to objects is certainly possible.

See Also

Overview and design of the `track` package.

`track.plugin.lru` *Plugins for cache policies in the track package.*

Description

Plugins for cache policies in the track package specify what objects should be keep in memory at the end of each top level command.

Usage

```
track.plugin.lru(objs, inmem, envname)
```

Arguments

`objs`	: the full object summary dataframe. A subset of this data frame is returned by `track.summary()`; invoke `track.summary(times=3, access=3,  size=T, all=T)` to get the full data frame. 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>`.

Details

`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.

`track.plugin.lru()` implements a simple least-recently-used discard policy. To use is, supply it to `track.options()`:

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

Here is another example of a very simple cache plugin function: this one keeps in memory variables whose names begin with the letter 'x'.

```
my.plugin <- function(objs, inmem, envname) {
    keep <- regexpr("^x", rownames(objs))>0
    # browser() # uncomment for debugging & development
    return(keep)
}
```

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

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

Value

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

Note

To flush cached tracked objects from memory, use `track.flush()`.

Author(s)

Tony Plate <tplate@acm.org>

`track.preremove` *Remove other resources associated with an object.*

Description

Remove other resources associated with an object prior to its removal by the tracking system. This S3 generic function exists so that methods can be specified for cleaning up particular objects.

Usage

```
track.preremove(obj, objName, envir, ...)
```

Arguments

`obj`	The R object that will be removed.
`objName`	The name of the R object that will be removed.
`envir`	The environment in which the object exists (i.e., the tracked environment, not the tracking environment.)
`...`	Other arguments.

Details

A `track.preremove()` method should clean up other resources (e.g., files) associated with `obj`. The `track.remove()` first calls `track.preremove()` and then will removes `obj` from the system after the call to `track.preremove()` has returned.

Value

Return values are ignored.

Author(s)

Tony Plate <`tplate@acm.org`>

See Also

track.remove

`track.rebuild` *Rebuild database information for tracked objects*

Description

Rebuild database information (the file map, and/or the object summary) for objects in an active tracking environment, or for saved objects in a tracking directory.

Usage

```
track.rebuild(pos = 1, envir = as.environment(pos), dir = NULL,
        fix = FALSE, level=c("missing", "all"),
        trust=c("unspecified", "environment", "db"),
        verbose=1, RDataSuffix=NULL, dryRun=TRUE, replace.wd=TRUE,
        use.file.times=TRUE)
```

Arguments

`pos`	The position of the environment to which the tracking database to rebuild is linked (for active tracking databases.
`envir`	The tracked environment for which to rebuild the tracking database (for active tracking databases.
`dir`	The directory where the tracking database is stored (for tracking databases not currently active.
`fix`	If `TRUE`, try to fix various problems such as illegal file names by renaming files or moving unusable RData files to a 'quarantine' directory. If `fix=TRUE` and `dryRun=TRUE`, changes will be reported but not made.
`level`	If `"missing"`, rebuild only missing information, if `"all"`, rebuild information for all objects.
`trust`	When rebuilding an active tracking database, and there are conflicts between the data in the environment versus that in files, which should be trusted?
`verbose`	Controls number of messages about progress
`RDataSuffix`	What suffix should be used for RData files? (Usually worked out automatically.)
`dryRun`	If `TRUE`, no data objects in R or files on disk are changed: the return value is a list containing the rebuilt file map and the rebuilt object summary. If `FALSE`, objects in R environments and files on disk can be changed. The default is `TRUE` to guard against changing things when it would have been better not to have changed things, but this default may change in future.
`replace.wd`	If `TRUE`, file paths in diagnostic output are printed starting with "./" where possible – this makes comparisons of test output more portable.
`use.file.times`	If `TRUE`, file creation, modification and access times are used for objects that were not found in existing summary objects. If `FALSE`, the current time is used, which can be useful for testing purposes.

Details

The file map and/or the object summary are rebuilt. If `level=="all"`, all RData files will be read, which could take a long time if there are many files. If `level=="missing"`, RData files will be read only where there is missing information.

If there are incompatible RData files in the directory (e.g., illegal or duplicated object names, or multiple objects), `track.rebuild` will stop with an error unless `fix==TRUE`, in which case the incompatible RData files will either be renamed or moved to a `quarantine` subdirectory. In the case of duplicated object names, the second object encountered will be moved.

Value

The value returned is a list with between two and four components:

`fileMap`	The mapping from object names to files (excluding the suffix)
`summary`	The dataframe that summarizes objects
`unsaved`	(optional) The names of variables that are not saved to disk. This component is only present if it is non-empty.
`masked`	(optinal) The names of tracked variables that are masked by other variables in the tracked environment. This component is only present if it is non-empty.

The returned value is invisible.

Author(s)

Tony Plate <tplate@acm.org>

See Also

Overview and design of the `track` package.

Examples

```
##############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
##############################################################

# Rebuild a damaged tracking database
library(track)
# first build a tracking dir populated with some variables
track.start(dir=file.path(tempdir(), 'rdatadir7'))
x <- 33
X <- array(1:24, dim=2:4)
Y <- list(a=1:3,b=2)
X[2] <- -1
```

```
abc <- "def"
def <- list(1,2,3)
invisible(Y); invisible(abc); invisible(abc); invisible(abc)
track.summary()
track.stop(pos=1)
# damage the database (remove the filemap)
unlink(file.path(tempdir(), 'rdatadir7', 'filemap.txt'))
# and rebuild
track.rebuild(dir=file.path(tempdir(), 'rdatadir7'), verbose=2, dryRun=FALSE, fix=TRUE)
track.start(file.path(tempdir(), 'rdatadir7'))
track.summary()
track.status()
# Would normally not call track.stop(), but do so here to clean up after
# running this example.
track.stop(pos=1, keepVars=TRUE)
```

track.rename *Rename variables in a tracked environment*

Description

Rename variables in a tracked environment

Usage

```
track.rename(old, new, pos = 1, envir = as.environment(pos),
             clobber = FALSE, verbose = TRUE)
```

Arguments

`old`	character vector of old names
`new`	character vector of new names (must be same length as 'old')
`pos`	The position on the search list of the tracked db. Can be numeric or the name as returned by `search()`.
`envir`	The tracked environment.
`clobber`	Should existing variables be overwritten?
`verbose`	Write out what is being done?

Details

Variables retain their tracked/untracked status (with the exception that a tracked variable can go from being tracked to being untracked if the new name matches the tracking option `autoTrackExcludePattern`.)

Renaming works correctly when old and new names overlap, e.g., `track.rename(c("x","y"), c("y","x"))` will swap the values of `x` and `y`.

Value

A list containing the old and new names:

`old`	a character vector of names
`new`	a character vector of names

Author(s)

Tony Plate <`tplate@acm.org`>

Examples

```
##############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
##############################################################

track.start(dir=file.path(tempdir(), 'rdatadir8'))
a <- 1
b <- rep(2, 2)
track.rename(c("a", "b"), c("b", "a"), clobber=TRUE)
c(a, b)
# Would normally not call track.stop(), but do so here to clean up after
# running this example.
track.stop(keepVars=TRUE)
```

`track.setup` *Setup and stop tracking*

Description

Functions to setup and stop tracking, and resync to a changed disk db

Usage

```
track.start(dir="rdatadir", pos = 1, envir = as.environment(pos),
        create = TRUE,
        clobber = c("no", "files", "variables", "vars", "var"),
        discardMissing = FALSE,
        cache = NULL, cachePolicy = NULL, options = NULL,
        RDataSuffix = NULL, auto = NULL, readonly = FALSE,
        lockEnv = FALSE, check.Last = TRUE, autoCheckSize = 1e6, verbose = TRUE)
track.stop(pos = 1, envir = as.environment(pos), all = FALSE,
        stop.on.error = FALSE, keepVars = FALSE, sessionEnd = FALSE,
        verbose = TRUE, detach = TRUE, callFrom = NULL)
track.rescan(pos = 1, envir = as.environment(pos), discardMissing = FALSE,
        forgetModified = FALSE, level = c("low", "high"), dryRun = FALSE,
        verbose = TRUE)
track.Last()
```

Arguments

`dir`	The directory where tracking data is stored
`pos`	The search path position of the environment being tracked (default is 1 for the global environment)
`envir`	The environment being tracked. This is an alternate way (to the use of `pos=`) of specifying the environment being tracked, but should be rarely needed.
`create`	If `TRUE`, create the tracking directory if it doesn't exist
`clobber`	Controls action taken when there are objects of the same name in the tracking directory and in `envir`: `no` means stop (unless objects are have the same values and are small); `files` means use the version from the tracking directory; and `variables`, `vars` or `var` means use the version in `envir` (and overwrite the version in the tracking directory). See also argument `autoCheckSize`.
`discardMissing`	Discard all information about objects whose save file is missing?
`cache`	Should objects be keep cached in memory? Default is `TRUE`. This option is a shorthand way of supplying `options=list(cache=...)`.
`cachePolicy`	Policy for keeping cached objects in memory. Default is `tltPurge`, for which cached objects are removed from memory at the end of a top-level task. This option is a shorthand way of supplying `options=list(cachePolicy=...)`.
`options`	Option values (as a list) to be used for this tracking database. See `track.options()`.
`all`	If `TRUE`, all tracked environment are unlinked
`auto`	If `TRUE`, automatically track new variables and deleted variables in the environment (through use of a task callback). If `auto==NULL`, take the value from `getOptions("global.track.options")$autoTrack`, and if that is `NULL`, use `TRUE`
`readonly`	Logical flag indicating whether the tracking db should be attached in a readonly mode. The global environment (`pos=1` in the search path) cannot be tracked in a readonly mode.
`stop.on.error`	If `FALSE`, failures to unlink a tracking environment are ignored, though a warning message is printed
`keepVars`	If `FALSE`, all tracked variables are removed and will be no longer accessible. If `TRUE`, tracked variables will be left as ordinary variables in the environment, as well as remaining in files.
`detach`	If `TRUE`, the environment attached to the search path (in a position other than 2) will be detached after stopping tracking, IF it was created by `track.attach()` and if there are no variables left remaining in the environment after removing

	all tracked variables. If `detach="force"`, the attached environment will be removed even if there are variables remaining in it (though not if it was not created by `track.attach`).
`callFrom`	A character string used in a message saying where `track.stop()` was called from.
`forgetModified`	If `TRUE`, discard the versions of objects that are modified and in memory
`RDataSuffix`	The suffix to use for `RData` files. This should not normally need to be specified.
`lockEnv`	Should the environment be locked for a readonly tracking environment? The default is `FALSE` because locking the environment is irreversible, and it prevents rescanning or caching (because can't delete or add bindings)
`check.Last`	By default, a warning is issued if the `.Last` function in the `track` package is masked by any other `.Last` function. Supplying `check.Last=FALSE` inhibits this check and warning.
`autoCheckSize`	If there are objects with the same name in an existing tracking db being attached, and the environment, and `clobber='no'`, `track.start()` will check whether the objects are the same (using `identical()`). If they are the same, `track.start()` will proceed. This check is only performed if the total size of the objects, and the files, is less than `autoCheckSize` bytes.
`level`	Should the rescan be done at a high or low level: a high level just stops tracking and restarts it; a low level tries to individually bring the environment in line with the file database.
`dryRun`	If `TRUE`, no changes are actualy made, but messages are printed describing what changes would be made.
`sessionEnd`	If `TRUE`, this is a call at the end of a session and no recovery from errors is possible – just try to as best as can to save objects as appropriate.
`verbose`	print a message about what directory is being tracked?

Details

`track.start`: start tracking `envir`. If the tracking directory already exists, objects in it will be made accessible, otherwise it will be created (unless `create=FALSE`).

`track.stop`: stop tracking `envir` (default is the global environment). Unsaved values will be saved to files first. Tracked variables will become unavailable unless `keepVars=TRUE` is supplied. If no arguments are supplied, stops tracking the global environment (pos=1). (In standard use, there is not a problem with only calling `track.stop()` prior to quitting R, thinking that it will cleanup all tracked environments, because tracked envs at positions other than 1 will be attached readonly.)

`track.rescan`: Rescan the tracking dir, so that if anything has changed there, the current variables on file will be used instead of any cached in memory. If we have some modified variables cached in memory but not saved to disk, this function will stop with an error unless `forgetModified==TRUE`. Variables that have disappeared from the tracking dir will disappear from visibility, and variables added to the tracking dir will become available.

.Last: `track.start()` or `track.attach()` set `.Last` in the global env to have the value `track.Last`, provided `.Last` does not already exist there. `.Last` will be called at the end of an R session, before the remaining variables in the global environment are saved to `.RData`. `track.Last` stops tracking all tracking db's, and removes tracked vars from their environments.

Value

`track.start, track.stop, track.rescan:`	all return `invisible(NULL)` (this may change if it becomes clear what useful return values would be)
`track.Last:`	calls `track.stop(all=TRUE)` to ensure that all tracking information and objects are written to files, and removes tracked variables from the environment.

Simple Usage

These functions have many arguments providing much control over tracking, but the arguments used in simple usage are:

```
track.start()
track.start(dir = "rdatadir")
track.stop(pos = 1, all = FALSE)
track.rescan(pos = 2)
```

Author(s)

Tony Plate <`tplate@acm.org`>

See Also

Overview and design of the `track` package.

Examples

```
###############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
###############################################################

library(track)
track.start(dir=file.path(tempdir(), 'rdatadir9'))
x <- 33
X <- array(1:24, dim=2:4)
Y <- list(a=1:3,b=2)
X[2] <- -1
track.datadir(relative=TRUE)
track.filename(list=c("x", "X"))
track.summary(time=0, access=1, size=FALSE)
env.is.tracked(pos=1)
env.is.tracked(pos=2)
ls(all=TRUE)
track.stop(pos=1)
ls(all=TRUE)
```

```
track.start(dir=file.path(tempdir(), 'rdatadir9'))
ls(all=TRUE)
track.summary(time=0, access=1, size=FALSE)
# Would normally not call track.stop(), but do so here to clean up after
# running this example.
track.stop(pos=1, keepVars=TRUE)
```

---

track.status *Return information about the status of tracking*

Description

Return information about the status of tracking in a particular environment. Functions tell which variables are and which are not tracked, and whether objects exist in memory or in files.

Usage

```
track.status(pos = 1, envir = as.environment(pos), expr,
        qexpr = NULL, list = NULL, pattern = NULL, glob = NULL,
        file.status = TRUE, tracked = NA, reserved = FALSE,
        all.names = FALSE,
        what = c("all", "tracked", "trackable", "untracked",
                 "orphaned", "masked", "unsaved", "untrackable"))
tracked(        pos=1, envir=as.environment(pos), list=NULL,
                pattern=NULL, glob=NULL, all.names = TRUE)
untracked(      pos=1, envir=as.environment(pos), list=NULL,
                pattern=NULL, glob=NULL, all.names = TRUE)
track.orphaned( pos=1, envir=as.environment(pos), list=NULL,
                pattern=NULL, glob=NULL, all.names = TRUE)
track.masked(   pos=1, envir=as.environment(pos), list=NULL,
                pattern=NULL, glob=NULL, all.names = TRUE)
untrackable(    pos=1, envir=as.environment(pos), list=NULL,
                pattern=NULL, glob=NULL, all.names = TRUE)
track.unsaved(  pos=1, envir=as.environment(pos), list=NULL,
                pattern=NULL, glob=NULL, all.names = TRUE)
```

Arguments

`pos`	The search path position of the environment being tracked (default is 1 for the global environment)
`envir`	The environment being tracked. This is an alternate way (to the use of pos=) of specifying the environment being tracked, but should be rarely needed.
`expr`	An unquoted variable name
`qexpr`	A variable name as an expression – not intended for use by end-users
`list`	A character vector of variable names
`pattern`	A regular expression specifying variable names to operate upon

`glob`	A glob pattern specifying variable names to operate upon
`file.status`	Check whether or not the file associated with a tracked object exists
`tracked`	If `TRUE`, return information only on tracked objects, if `FALSE`, return information only on objects that are not tracked, and if `NA` return information on all variables (subject to other filtering).
`all.names`	should names beginning with a period be included (like `all.names` in `ls`)
`what`	controls the information returned: `"all"` means return a data frame of status, other values means return a list of names of objects having that status
`reserved`	If `TRUE`, include info about non-tracked variables with reserved names. The default is to always omit these variables from the status summary.

Details

These functions return information about the status of tracking on some or all variables in `envir` and the tracking environment. Tracking status depends on the relationship among four entities used for a tracked object:

- the name of the object
- the binding in `envir` which should be an active binding that refers to the tracking environment
- the cached object in the tracking environment (i.e., stored in memory in R)
- the corresponding disk file in the tracking directory

Statuses are defined as follows:

Status	object name	variable	cached object	file
tracked	ordinary	active binding	maybe	yes, maybe up-to-date
untrackable	reserved name	ordinary	no	no
untracked	ordinary	ordinary	no	no
masked	ordinary	ordinary	yes	maybe
orphaned	ordinary	none	yes	maybe
unsaved	ordinary	active binding	yes	not up-to-date

The arguments `expr`, `list`, `pattern`, and `glob` all serve to restrict the set of variables considered.

Value

`track.status:`	returns a `data.frame` if `what=="all"`, or a character vector otherwise.
`track.dir:`	returns a single character string that is the full path to the directory where copies of objects are stored (the "tracking directory").
`tracked, untracked, track.orphaned, track.masked, untrackable, track.unsaved:`	all return a character vector naming the variables with a particular status.

Note

These functions check whether the binding in `envir` is an active binding, but they cannot check whether the active binding has the correct function associated with it because R provides no mechanism for R-level access to the function associated with active bindings.

Author(s)

Tony Plate <tplate@acm.org>

See Also

Overview and design of the `track` package.

Examples

```
###############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
###############################################################

library(track)
track.start(dir=file.path(tempdir(), 'rdatadir10'))
x1 <- 123
x2 <- 456
x3 <- 789
track.status()
rm(x3)
track.status()
# Would normally not call track.stop(), but do so here to clean up after
# running this example.
track.stop(pos=1, keepVars=1)
```

`track.summary` *Return a summary of the basic properties of tracked objects*

Description

Return a summary of the basic properties of tracked objects: name, class, size, dimensions (if any), and creation, modification and access times.

Usage

```
track.summary(expr, pos = 1, envir = as.environment(pos), list = NULL, pattern = NULL,
         glob = NULL, all.names = FALSE,
         times = track.options("summaryTimes", envir=envir)[[1]],
         access = track.options("summaryAccess", envir=envir)[[1]],
         size=TRUE, cache=FALSE, full=FALSE)
```

Arguments

`expr`	: An unquoted variable name
`pos`	: The search path position of the environment being tracked (default is 1 for the global environment)
`envir`	: The environment being tracked. This is an alternate way (to the use of `pos=`) of specifying the environment being tracked, but should be rarely needed.
`list`	: A character vector of variable names to summarize
`pattern`	: A regular expression specifying variable names to summarize
`glob`	: A regular expression specifying variable names to summarize
`all.names`	: should names beginning with a period be included (like `all.names` in `ls`)
`times`	: An integer 0..3 specifying how many time columns to return (in order of: modify, create, access)
`access`	: An integer 0..3 specifying how many access-count columns to return for each writes and accesses (0=none, 1=total, 2=prior and current session, 3=prior, current session and total)
`size`	: (logical) include the `size` column? (The values in this column are system dependent, so make it easy to exclude so that test output is constant across systems.)
`cache`	: (logical) include the `cache` column?
`full`	: (logical) include all the optional columns?

Details

Returns part or all of the cached summary data. There is one row per object. Only tracked objects appear in the summary.

Value

The value returned is a dataframe that summarizes the specified objects. This function does not create any output itself – the auto-printing of the returned value is the visible output. The data frame has one row for each object (rownames are the object names) and some of the following columns:

`class:`	(character) from `class()` (if `class(obj)` has more than one component, all components are appended separated by commas, if `class(obj)` returns a zero-length result, the value is `"?"`)
`mode:`	(character) from `mode()`
`extent:`	(character) from `dim()` or `length()`, with double brackets if the object is a list (will contain `"(error)"` if `dim(obj)` causes an error)
`length:`	(integer) from `length()` (will be `NA` if `length(obj)` causes an error)
`size:`	(integer) from `object.size()`
`cache:`	(character): Indicates whether the objected should always be kept cached. Possible values are

yes, no: Automatically determined from `track.options("alwaysCache")` (object names) or `track.options("alwaysCacheClass")` (object classes); re-evaluated when object changes

fixedyes, fixedno: Always do this; don't re-evaluate when object changes. This is intended to allow manually setting this for each object if desired, but as of version 1.0.9 no way for the user to do this safely has been provided.

`modified:`	most recent modification time
`created:`	time object created
`read:`	most recent modification time
`A:`	(logical) Accuracy of times: `TRUE` if object summary has been maintained since object was first tracked; `FALSE` if the object summary was lost at some point and then recreated
`ES:`	(integer) sessions alive
`SR:`	(integer) num reads this session
`SW:`	(integer) num writes this session
`PR:`	(integer) total reads prior to this session
`PW:`	(integer) total writes prior to this session
`TR:`	(integer) total reads
`TW:`	(integer) total writes

Which columns are present depends on the arguments `times`, `access`, and `size`.

The reason for the `class` column containing all classes of the object separated by commas is that extracting the most informative class label is not simple, for example, the class of an object returned by `glm()` is `c("glm", "lm")` (most informative first), while the class of an object returned by `Sys.time()` is `c("POSIXt", "POSIXct")` (most informative last).

Note

The object summary data is maintained in an object called `.trackingSummary` kept in the tracking environment. It is not visible on the search path.

Author(s)

Tony Plate <tplate@acm.org>

See Also

Overview and design of the `track` package.

Examples

```
##############################################################
# Warning: running this example will cause variables currently
# in the R global environment to be written to .RData files
# in a tracking database on the filesystem under R's temporary
# directory, and will cause the variables to be removed temporarily
# from the R global environment.
# It is recommended to run this example with a fresh R session
# with no important variables in the global environment.
##############################################################

library(track)
track.start(dir=file.path(tempdir(), 'rdatadir11'))
x <- 33
X <- array(1:24, dim=2:4)
Y <- list(a=1:3,b=2)
X[2] <- -1
y1 <- 2
y2 <- 3
track.summary()
track.summary(time=0, access=1, size=FALSE)
track.summary(X)
track.summary(list=c("x", "X"))
track.summary(pattern="[xX]")
# Would normally not call track.stop(), but do so here to clean up after
# running this example.
track.stop(pos=1, keepVars=TRUE)
```

`track.sync` *Synchronize the tracking database stored in files on disk to reflect changed, new, or deleted objects in R, and flush excess objects from memeory.*

Description

Synchronize the tracking database to reflect new and/or deleted objects. This function is intended to be called by a task callback so that the tracking database automatically keeps up with new and deleted objects. The appropriate task callback is installed by `track.start(..., auto = TRUE)`. If too much memory is occupied by objects, excess objects are flushed from memory.

This function differs from `track.rescan()` in that `track.rescan()` updates R view of the database to agree with changes on disk, while `track.sync()` is primarily intended to go the other way (to make the disk database agree with R).

Usage

```
track.sync(pos = 1, master=c("auto", "envir", "files"), envir = as.environment(pos),
            trackingEnv = getTrackingEnv(envir), full = TRUE,
            dryRun = FALSE, taskEnd = FALSE)
```

Arguments

`pos`	The search path position of the environment being tracked (default is 1 for the global environment)

`master`	What to treat as the master for the synchronization. For a readonly tracked environment, the default `auto` will default to `"files"` (the only sensible interpretation). Otherwise, must be supplied (to avoid accidents with users misremembering the default.) When `master="envir"`, changes are propagated from the R environment to the file system. When `master="files"`, the R environment is made to reflect the file system (differences could result from changes to the R environment, or from changes to the file system, e.g., by another R process changing the database).
`envir`	The environment being tracked. This is an alternate way (to the use of `pos=`) of specifying the environment being tracked, but should be rarely needed.
`trackingEnv`	The environment that contains data for the tracking database.
`full`	If `TRUE`, do a full check, which involves checking that all apparently tracked variables do in fact have an active binding. If `NA`, only do a full check if more than `track.options("autoTrackFullSyncWait")` seconds have passed since the last full check (because this check can be slow when there are many variables in the environment.
`dryRun`	If `TRUE`, no changes are made to either the file system or to the R environment, but changes that would be made are printed out. Note that a change to a file for a tracked variable will not be detect.
`taskEnd`	Should be `TRUE` when called at the end of a top-level command task (i.e., when called by the task callback handler.)

Details

Synchronizing the tracking database with the contents of the environment involves three tasks:

1. start tracking new untracked variables
2. for objects that have disappeared from the environment, delete them from the tracking database
3. check that all apparently tracked variables do in fact have an active binding

Currently, this function will not correctly handle the case where `master="files"` and where objects are cached and an underlying file is changed.

Value

Returns an invisible list with the following components:

`new`	character vector of names of new variables
`removed`	character vector of names of variables that were removed

Note

The check that all apparently tracked variables have an active binding currently only checks that the variable has an active binding – there is no way (at the R level) to check that the active binding is the correct one.

Author(s)

Tony Plate <tplate@acm.org>

See Also

`track.rescan` for rescanning a tracked database after the files on disk have changed (this is usually only used for tracked environment attached at a position 2 or greater.)

Overview and design of the `track` package.

Index

∗Topic **database**
    track-intro, 3
∗Topic **data**
    track-intro, 3
    track.attach, 10
    track.auto, 11
    track.copy, 12
    track.design, 14
    track.future, 19
    track.info, 24
    track.manage, 26
    track.options, 29
    track.performance, 33
    track.plugin.lru, 34
    track.preremove, 36
    track.rebuild, 37
    track.rename, 39
    track.setup, 40
    track.status, 44
    track.summary, 46
    track.sync, 49
∗Topic **misc**
    show.envs, 2
    track.history, 22
∗Topic **package**
    track-intro, 3
∗Topic **utilities**
    track-intro, 3

addTaskCallback, *9*, *23*

Design, *9*
design, *21*, *25*, *28*, *32*, *34*, *38*, *43*, *46*, *48*, *51*

env.is.tracked, *7*
env.is.tracked *(*track.info*)*, 24

filehash, *9*, *19*
future features, *9*

g.data, *9*, *19*

load, *9*
loadhistory, *23*

makeActiveBinding, *9*, *19*

Overview, *19*, *21*, *25*, *26*, *28*, *32*, *34*, *38*, *43*, *46*, *48*, *51*

save, *9*, *31*
savehistory, *23*
show.envs, 2, *8*

taskCallback, *7*
track, *5*, *6*
track *(*track.manage*)*, 26
track-intro, 3
track-overview *(*track-intro*)*, 3
track-package *(*track-intro*)*, 3
track.attach, *5*, 10
track.auto, *7*, 11
track.cache *(*track.plugin.lru*)*, 34
track.copy, *7*, 12
track.datadir *(*track.info*)*, 24
track.design, *8*, 14
track.detach *(*track.attach*)*, 10
track.dir, *6*
track.dir *(*track.status*)*, 44
track.filename *(*track.info*)*, 24
track.flush, *7*, *34*, *35*
track.forget, *7*
track.future, 19
track.history, 22
track.info, *5*, *6*, 24
track.intro *(*track-intro*)*, 3
track.Last *(*track.setup*)*, 40
track.load, *6*, *7*
track.manage, 26
track.masked, *7*
track.masked *(*track.status*)*, 44
track.mem, *6*

track.mem *(*track.info*)*, 24
track.move, *7*
track.move *(*track.copy*)*, 12
track.options, *4*, *7*, *15*, 29, *34*
track.orphaned, *7*
track.orphaned *(*track.status*)*, 44
track.overview *(*track-intro*)*, 3
track.package *(*track-intro*)*, 3
track.performance, 33
track.plugin *(*track.plugin.lru*)*, 34
track.plugin.lru, 34
track.plugins, *31, 32*
track.plugins *(*track.plugin.lru*)*, 34
track.preremove, 36
track.rebuild, *8*, 37
track.remove, *6*, *7*, *36*
track.rename, *7*, 39
track.rescan, *5*, *7*, *49*, *51*
track.rescan *(*track.setup*)*, 40
track.save, *7*, *34*
track.setup, 40
track.start, *4–6*, *10–12*
track.start *(*track.setup*)*, 40
track.status, *6*, 44
track.stop, *5*, *6*, *11*
track.stop *(*track.setup*)*, 40
track.summary, *5*, *6*, *31*, 46
track.sync, *7*, 49
track.unsaved, *7*
track.unsaved *(*track.status*)*, 44
tracked, *7*
tracked *(*track.status*)*, 44
tracked.envs *(*track.info*)*, 24

untrack, *6*
untrack *(*track.manage*)*, 26
untrackable, *7*
untrackable *(*track.status*)*, 44
untracked, *7*
untracked *(*track.status*)*, 44