Package: context (via r-universe)

June 19, 2024

Title Contexts for evaluating R expressions

Version 0.5.0

Description Contexts for evaluating R expressions.

License MIT + file LICENSE

LazyData true

URL https://github.com/mrc-ide/context

BugReports https://github.com/mrc-ide/context/issues

Depends R (>= 3.2.0)

Imports crayon, ids, storr (>= 1.1.1)

Suggests callr, knitr, mockery, parallel, redux, rmarkdown, testthat, withr

RoxygenNote 7.2.3

Encoding UTF-8

VignetteBuilder knitr

Language en-GB

Repository https://mrc-ide.r-universe.dev

RemoteUrl https://github.com/mrc-ide/context

RemoteRef master

RemoteSha 62dbb46b1c70c4cfa1404d2f5ecbe6c4933f226c

Contents

`bulk_prepare_expression` *Prepare many expressions*

Description

Prepare many expressions

Usage

```
bulk_prepare_expression(X, FUN, DOTS, do_call, use_names, envir, db)
```

Arguments

`X`	Something to iterate over; a vector, list or data.frame (in the case of a data.frame, iteration will be row-by-row)
`FUN`	A function to apply to each element (or row) of `X`
`DOTS`	Additional arguments to apply with each elements of `X`
`do_call`	Treat each element of `X` as a `do.call` call
`use_names`	When preparing a data.frame, retain column names as argument names when using `do_call`. If `FALSE` then positional matching will be used.
`envir`	An environment to find variables local to the expression
`db`	A database to store locals

bulk_task_save *Save bulk tasks*

Description

Save bulk tasks

Usage

```
bulk_task_save(
  X,
  FUN,
  context,
  DOTS = NULL,
  do_call = FALSE,
  use_names = TRUE,
  envir = parent.frame(),
  depends_on = NULL
)
```

Arguments

X	Something to iterate over; a vector, list or data.frame (in the case of a data.frame, iteration will be row-by-row)
FUN	A function to apply to each element (or row) of X
context	A context
DOTS	Additional arguments to apply with each elements of X
do_call	Treat each element of X as a do.call call
use_names	When preparing a data.frame, retain column names as argument names when using do_call. If FALSE then positional matching will be used.
envir	An environment to find variables local to the expression
depends_on	Optional task ids that this task depends on. To have all tasks depend on the same id(s) provide a vector. TO provide different dependencies for each task provide a list of lists. For example list(list("abcde", "12345"), list(), list("12345"))

context_list *List save contexts*

Description

List saved contexts

Usage

```
context_list(db, named = FALSE, error = TRUE)

context_info(db, error = TRUE)
```

Arguments

db	Something for which a context database can be created; this can the the path to the context, a `context_root` object, or a `context` object.
named	Logical, indicating if the context name should be used to name the output vector.
error	Throw an error if the context database cannot be connected constructed (e.g., if the path given does not exist).

Author(s)

Rich FitzJohn

---

context_load *Load a context*

Description

Load a context

Usage

```
context_load(ctx, envir = .GlobalEnv, refresh = FALSE)
```

Arguments

ctx	A context object, as read by `context_read`
envir	The environment to source files into
refresh	Refresh the context, even when it has been loaded already? Note that this may not always behave as expected because items not created by sourcing an R file will still be there from previous runs, and packages loaded will not be reloaded in a new order.

context_log *Send entry to context log*

Description

Send an entry to the context log. This is designed primarily for use with packages that build off of context, so that they can log in a consistent way.

Usage

```
context_log(topic, value)
```

Arguments

topic	Up to 9 character text string with the log topic
value	Character string with the log entry

context_read *Read a context*

Description

Read a context

Usage

```
context_read(identifier, root, db = NULL)
```

Arguments

identifier	Either the id or name of a context (see context_list)
root	Something interpretable as the context root; either
db	Optionally, a database (if known already)

context_root_get *Find context root*

Description

Find the context root. Designed for internal use

Usage

```
context_root_get(root, db = NULL)
```

Arguments

root	An object; either a character string (interpreted as a path), a context_root object (such as returned by this function) or a list/environment object with a root element that is a context_root object.
db	Optionally, a copy of the storr database (if already opened). Do not specify this unless you *definitely* have the correct database in hand.

context_save *Save a context*

Description

Save a context

Usage

```
context_save(
  path,
  packages = NULL,
  sources = NULL,
  package_sources = NULL,
  envir = NULL,
  storage_type = NULL,
  storage_args = NULL,
  name = NULL,
  root_id = NULL
)
```

Arguments

path	Path to save the context in
packages	Optional character vector of packages to save into the context. Alternatively, can be a list with elements loaded and attached if you want to ensure some packages are loaded but not attached.
sources	Character vector of source files to read in. These should define functions and (perhaps) other "global" objects, but should not do any serious computation.
package_sources	Optional information about where to find non-CRAN packages, created by conan::conan_sources
envir	The current environment. This is used to copy *local* variables around. For context_load this is the environment into which the global environment is copied. Specify a non-global environment here to avoid clobbering the workspace, but at the risk that some environments may not restore exactly as desired. If this is used, then every new R session, running context_save will create a new context id.
storage_type	Character vector indicating the storage type to use. Options are "rds" (the default) and "environment" (for testing and local use).
storage_args	Arguments passed through to the storage driver
name	An optional name for the context. This will be printed with the context in some situations (such as context_info)
root_id	Force a context root id. This is intended for advanced use only. By setting the root id, two contexts created with storage in different file locations (path) will get the same id. This is required for using a server-hosted database to share a context between different physical machines (or different docker containers). The id, if provided, must be compatible with ids::random_id() - i.e., a 32 character hex string. This option can be left alone in most situations.

---

last_loaded_context *Return last loaded context*

Description

Return the last loaded context

Usage

```
last_loaded_context(error = TRUE)
```

Arguments

error	Throw an error if no context has been loaded

parallel_cluster_start *Start a sub-cluster*

Description

Start a sub-cluster, using the `parallel` package. This will be available via either the return value of this function, the `parallel_cluster` function or by using `cl = NULL` with any of the `parallel` package functions. The cluster will be started so that it is ready to use the context.

Usage

```
parallel_cluster_start(n, ctx)

parallel_cluster_stop()

parallel_cluster()
```

Arguments

`n`	The number of nodes. No attempt at guessing this number is made as that is terribly error prone. If you're using this function you should know how many resources you have available.
`ctx`	The context to initialise on each cluster node.

prepare_expression *Prepare expression*

Description

Prepare expression for evaluation in context

Usage

```
prepare_expression(expr, envir, db, function_value = NULL)
```

Arguments

`expr`	A quoted expression consisting of a single function call.
`envir`	An environment to find variables local to the expression
`db`	A database to store locals
`function_value`	Optionally, the *value* of a function where the expression should involve an anonymous function. In this case the function in `expr` will be replaced.

Details

The function_value argument here is used where expr is going to take a function that is not addressable by *name*; in that case we take a function itself (as "function_value"), serialise it and replace the function call with the hash. The function will be serialised into the calling environment on deserialisation.

This includes the remote possibility of a collision, but with the size of the keyspace used for hashes hopefully it's negligible.

Because of the approach used here, expr can contain anything; I'd suggest not saving the contents of the function itself, but something like NULL will work just fine:

```
as.call(list(NULL, quote(a)))
# NULL(a)
```

---

restore_locals *Restore locals*

Description

Restore locals created by prepare_expression.

Usage

```
restore_locals(dat, parent, db)
```

Arguments

dat	An expression that has been through prepare_expression. Key elements are function_hash and objects
parent	The parent environment to restore locals to
db	The database used to prepare the expression

---

task_context_id *Find context for a task*

Description

Find the context id associated with a task

Usage

```
task_context_id(ids, db)
```

Arguments

ids	Vector of task ids
db	Something that can be converted to a context db object (a database, root or context).

task_delete *Delete a task*

Description

Delete a task, including its results.

Usage

```
task_delete(ids, root)
```

Arguments

ids	Vector of task ids
root	A context root (not just the db as in task_result as we need to know the actual path to the root). A context object is also OK.

Value

TRUE if a task was actually deleted.

task_deps *Task dependencies*

Description

Task dependencies

Usage

```
task_deps(ids, db, named = FALSE)
```

Arguments

ids	Vector of task ids
db	Something that can be converted to a context db object (a database, root or context).
named	Name the output with the task ids?

task_exists *List tasks*

Description

List tasks and test if they exist

Usage

```
task_exists(ids, db)

task_list(db)
```

Arguments

ids	Vector of task ids
db	Something that can be converted to a context db object (a database, root or context).

task_expr *Fetch task expression*

Description

Fetch expression for a task

Usage

```
task_expr(id, db, locals = FALSE)
```

Arguments

id	Single task identifier
db	Something that can be converted to a context db object (a database, root or context).
locals	Return locals bound to the expression (as an attribute "locals")

task_function_name *Fetch task function name*

Description

Fetch function name for a task

Usage

```
task_function_name(ids, db)
```

Arguments

ids	Vector of task ids
db	Something that can be converted to a context db object (a database, root or context).

task_log *Return task log*

Description

Return the log of a task, if enabled.

Usage

```
task_log(id, root, parse = TRUE)
```

Arguments

id	Single task identifier
root	A context root (not just the db as in task_result as we need to know the actual path to the root). A context object is also OK.
parse	Parse the log output into a context_log object, which will pretty print and can be more easily inspected. If FALSE then the raw log will be returned as a character vector, one element per line of text

Details

The returned object is of class task_log, which has a print method that will nicely display. Output is grouped into phases.

task_reset *Reset status and submission time of tasks*

Description

Reset tasks

Usage

```
task_reset(id, context)
```

Arguments

id	A vector of task identifiers
context	A context object

task_result *Fetch task result*

Description

Fetch result from completed task.

Usage

```
task_result(id, db, allow_incomplete = FALSE)
```

Arguments

id	Single task identifier
db	Something that can be converted to a context db object (a database, root or context).
allow_incomplete	Should we avoid throwing an error if a task is not completed? Used internally, and not generally needed.

task_run *Run a task*

Description

Run a task

Usage

```
task_run(id, context, filename = NULL)
```

Arguments

id	A task identifier
context	A context object
filename	Filename to log *all* output to. This will sink the message stream as well as the output stream, so if specified (i.e., is non-NULL) then this function will apparently print no output to the console, which will make debugging more difficult when run interactively. However, when run non-interactively, especially on remote servers, this will allow collection of diagnostics that facilitate debugging.

task_run_external *Run a task in separate process*

Description

Run a task in a separate process, using [callr::r]. Unlike [context::task_run] this does not return the value, and is called for the side effect of writing to the context.

Usage

```
task_run_external(root, identifier, task_id, path_log)
```

Arguments

root	Something interpretable as the context root; either
identifier	Either the id or name of a context (see context_list)
task_id	A task identifier
path_log	Path to log file

task_save *Save and reload tasks*

Description

Save and reload tasks. Tasks consist of an expression bound to a `context`.

Usage

```
task_save(expr, context, envir = parent.frame(), depends_on = NULL)
```

Arguments

`expr`	An expression to save
`context`	A `context` object
`envir`	Passed through to `context_save` when locating local variables.
`depends_on`	Optional vector of task ids that this task depends on

Value

An identifier that can be used to retrieve or run the task later. This is simply a short string.

task_status *Task status*

Description

Task status

Usage

```
task_status(ids, db, named = FALSE)
```

Arguments

`ids`	Vector of task ids
`db`	Something that can be converted to a context db object (a database, root or context).
`named`	Name the output with the task ids?

`task_times` *Fetch task times*

Description

Fetch times taken to queue, run, and since running a task.

Usage

```
task_times(ids, db, unit_elapsed = "secs", sorted = TRUE)
```

Arguments

`ids`	Vector of task ids
`db`	Something that can be converted to a context db object (a database, root or context).
`unit_elapsed`	Elapsed time unit. The default is "secs". This is passed to the `as.numeric` method of a `difftime` object.
`sorted`	Sort the output in terms of submitted time? If `FALSE` then the output is sorted based on task ids.

Author(s)

Rich FitzJohn

Index