

# Package: context (via r-universe)

September 17, 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 . . . . .	2
bulk_task_save . . . . .	3
context_list . . . . .	3
context_load . . . . .	4
context_log . . . . .	5
context_read . . . . .	5
context_root_get . . . . .	6

context_save . . . . .	6
last_loaded_context . . . . .	7
parallel_cluster_start . . . . .	8
prepare_expression . . . . .	8
restore_locals . . . . .	9
task_context_id . . . . .	9
task_delete . . . . .	10
task_deps . . . . .	10
task_exists . . . . .	11
task_expr . . . . .	11
task_function_name . . . . .	12
task_log . . . . .	12
task_reset . . . . .	13
task_result . . . . .	13
task_run . . . . .	14
task_run_external . . . . .	14
task_save . . . . .	15
task_status . . . . .	15
task_times . . . . .	16
<b>Index</b>	<b>17</b>

---

bulk_prepare_expression
<i>Prepare many expressions</i>

---

**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	<i>Save bulk tasks</i>
----------------	------------------------

---

**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	<i>List save contexts</i>
--------------	---------------------------

---

**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 be 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	<i>Load a context</i>
--------------	-----------------------

---

**Description**

Load a context

**Usage**

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

**Arguments**

ctx	A context object, as read by <a href="#">context_read</a>
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	<i>Send entry to context log</i>
-------------	----------------------------------

---

**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	<i>Read a context</i>
--------------	-----------------------

---

**Description**

Read a context

**Usage**

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

**Arguments**

identifier	Either the id or name of a context (see <a href="#">context_list</a> )
root	Something interpretable as the context root; either
db	Optionally, a database (if known already)

---

context_root_get	<i>Find context root</i>
------------------	--------------------------

---

### 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 <i>definitely</i> have the correct database in hand.

---

context_save	<i>Save a context</i>
--------------	-----------------------

---

### 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 <code>conan::conan_sources</code>
envir	The current environment. This is used to copy <i>local</i> variables around. For <code>context_load</code> 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 <code>context_save</code> 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 <code>context_info</code> )
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 <code>ids::random_id()</code> - i.e., a 32 character hex string. This option can be left alone in most situations.

---

last_loaded_context	<i>Return last loaded context</i>
---------------------	-----------------------------------

---

**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 <i>value</i> 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	<i>Restore locals</i>
----------------	-----------------------

---

Description

Restore locals created by [prepare\\_expression](#).

Usage

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

Arguments

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

---

task_context_id	<i>Find context for a task</i>
-----------------	--------------------------------

---

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	<i>Delete a task</i>
-------------	----------------------

---

**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 <a href="#">task_result</a> 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	<i>Task dependencies</i>
-----------	--------------------------

---

**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	<i>List tasks</i>
-------------	-------------------

---

**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	<i>Fetch task expression</i>
-----------	------------------------------

---

**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	<i>Fetch task function name</i>
--------------------	---------------------------------

---

### 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	<i>Return task log</i>
----------	------------------------

---

### 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 <a href="#">task_result</a> 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	<i>Reset status and submission time of tasks</i>
------------	--

---

**Description**

Reset tasks

**Usage**

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

**Arguments**

id	A vector of task identifiers
context	A context object

---

task_result	<i>Fetch task result</i>
-------------	--------------------------

---

**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	<i>Run a task</i>
----------	-------------------

---

**Description**

Run a task

**Usage**

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

**Arguments**

id	A task identifier
context	A context object
filename	Filename to log <i>all</i> 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	<i>Run a task in separate process</i>
-------------------	---------------------------------------

---

**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 <a href="#">context_list</a> )
task_id	A task identifier
path_log	Path to log file

---

task_save	<i>Save and reload tasks</i>
-----------	------------------------------

---

**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	<i>Task status</i>
-------------	--------------------

---

**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	<i>Fetch task times</i>
------------	-------------------------

---

**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 <code>as.numeric</code> method of a <code>difftime</code> 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

bulk\_prepare\_expression, [2](#)  
bulk\_task\_save, [3](#)

context\_info, [7](#)  
context\_info(context\_list), [3](#)  
context\_list, [3](#), [5](#), [14](#)  
context\_load, [4](#)  
context\_log, [5](#)  
context\_read, [4](#), [5](#)  
context\_root\_get, [6](#)  
context\_save, [6](#)

last\_loaded\_context, [7](#)

parallel\_cluster  
    (parallel\_cluster\_start), [8](#)  
parallel\_cluster\_start, [8](#)  
parallel\_cluster\_stop  
    (parallel\_cluster\_start), [8](#)  
prepare\_expression, [8](#), [9](#)

restore\_locals, [9](#)

task\_context\_id, [9](#)  
task\_delete, [10](#)  
task\_deps, [10](#)  
task\_exists, [11](#)  
task\_expr, [11](#)  
task\_function\_name, [12](#)  
task\_list(task\_exists), [11](#)  
task\_log, [12](#)  
task\_reset, [13](#)  
task\_result, [10](#), [12](#), [13](#)  
task\_run, [14](#)  
task\_run\_external, [14](#)  
task\_save, [15](#)  
task\_status, [15](#)  
task\_times, [16](#)