Package: sxpdb (via r-universe)

July 15, 2026

Title Store R values

Version 0.1

Description Stores R values. It stores the values in an efficient ways that scales to millions of unique values and provides a rich query API to find values from the database.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

SystemRequirements C++17

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports pkgload

Suggests testthat, xml2, mutator

LinkingTo testthat

Remotes PRL-PRG/mutator

Config/pak/sysreqs cmake make libuv1-dev

Repository https://prl-prg.r-universe.dev

Date/Publication 2026-07-15 21:17:10 UTC

RemoteUrl https://github.com/PRL-PRG/sxpdb

RemoteRef HEAD

RemoteSha eacdf11be92bb7dbb636fadd6de641e6ffe9c0bc

Contents

`add_origin` *Add origin to an already recorded value*

Description

`add_origin` adds package, function and argument names for a value already recorded in the database. The value is represented here by its hash. You will not probably use it and rather use `add_val_origin()` directly, as `add_origin` will only save you the hash computation. If performance is an issue, you should not use the R API anyway and directly hook into the C one.

Usage

```
add_origin(db, hash, package, func, argument)
```

Arguments

db	database, sxpdb object
hash	hash of an already recorded value
package	character vector for the package name
func	character vector for the function name
argument	character vector for the argument name. "" or NA mean that val is a return value.

Value

NULL

See Also

add_val_origin(), build_indexes()

---

add_val *Add a value in the dabatase*

Description

add_val adds an R value in the database. It does not add origins or call ids. This should be used in conjunction with add_origin(). You should most likely never use it and rather directly use add_val_origin().

Usage

```
add_val(db, val)
```

Arguments

db	database, sxpdb object
val	any R value. Currently, environments and closures will be silently ignored

Value

integer index of the value if it has been added (now or before), or NULL if it has been ignored

See Also

add_val_origin(), add_origin(), build_indexes()

add_val_origin *Add a value with origin and call id*

Description

`add_val_origin` adds an R value in the database, along with its origin (package, function, argument names), and its call id.

Usage

```
add_val_origin(db, val, package, func, argument, call_id = 0)
```

Arguments

`db`	database, sxpdb object
`val`	any R value. Currently, environments and closures will be silently ignored
`package`	character vector for the package name
`func`	character vector for the function name
`argument`	character vector for the argument name. `""` or `NA` mean that `val` is a return value.
`call_id`	integer unique id of the call from which the value comes from. Defaults to `0`

Value

integer index of the value if it has been added (now or before), or `NULL` if it has been ignored

See Also

`build_indexes()`

build_indexes *Build search indexes.*

Description

`build_indexes` explicitly builds search indexes which are used by function with a `query` argument (when it is not `NULL`). You need to use it if you add new values into the database (including merging into it). However, `merge_all_dbs()` will automatically build the search indexes.

Usage

```
build_indexes(db)
```

Arguments

db	database, sxpdb object

Value

NULL

---

check_all_db *Checks if the db is not corrupted.*

Description

check_all_db checks if the database is not currupted and suggests some fixes if it is.

Usage

```
check_all_db(db, slow = FALSE)
```

Arguments

db	database, sxpdb object
slow	boolean, if TRUE, it will unserialize all the values in the database and check them all, if 'FALSE, it will perform only quick consistency checks.

Value

integer vector of indices of values with problems

---

close_db *Closes the database*

Description

Closes a previously open database. If db was uncorrectly opened, or has been already closed, the function will error.

Usage

```
close_db(db)
```

Arguments

db	database, sxpdb object

Value

NULL

See Also

open_db()

---

close_query *Closes a query object.*

Description

close_query closes a query object. A query object is implemented as an external pointer to a a C++ object that he dynamically allocated. However, a GC hook for that external pointer is also registered so except if you plan to crate a huge amount of query objects, you should be fine with not calling close_query explicitly.

Usage

```
close_query(query)
```

Arguments

query	query object

Value

NULL

See Also

query_from_value(), query_from_plan(), relax_query()

---

filter_index_db *Filters values from the database according to some predicate*

Description

Sometimes, the query language is not enough to select some values. You can use filter_index_db to refine a query and only keep the values for which the predicate is true.

Usage

```
filter_index_db(db, fun, query = NULL)
```

Arguments

db	database, sxpdb object
fun	R function, the predicate, which should take one argument, the value, and return a boolean. NA_logical_is considered asTRUE‘.
query	query object, typically built from query_from_plan() or query_from_value().

Value

list of indices of the values matche2d by the query for which `fun` evaluated to `TRUE` or `NA_logical_`.

See Also

`map_db()`, `view_db()`

`get_meta` *Get the metadata associated to a value*

Description

`get_meta` returns the metadata associated to a value. It includes: * *runtime* metadata: number of calls the value has been seen in, number of times the value has been seen during merges * *static* metadata: type, size in bytes, length (for vector values), number of attributes, number of dimensionsm number of rows (for data frames, matrixes) * *debug* metadata: only if the database was created with sxpdb in debug mode, includes how many times `MAYBE_SHARED` has been true on the value, and how many times we were able to use the SEXP address optimization

Usage

```
get_meta(db, val)
```

Arguments

`db`	database, sxpdb object
`val`	a R value

Value

a named list with the metadata, or `NULL` if there is no such value in the database

See Also

`get_meta_idx()`

get_meta_idx *Get the metadata associated to an index*

Description

`get_meta` returns the metadata associated to a index in the database. It includes: * *runtime* metadata: number of calls the value has been seen in, number of times the value has been seen during merges * *static* metadata: type, size in bytes, length (for vector values), number of attributes, number of dimensionsm number of rows (for data frames, matrixes) * *debug* metadata: only if the database was created with sxpdb in debug mode, includes how many times `MAYBE_SHARED` has been true on the value, and how many times we were able to use the SEXP address optimization

Usage

```
get_meta_idx(db, idx)
```

Arguments

`db`	database, sxpdb object
`idx`	integer index of the value to look for

Value

a named list with the metadata, or `NULL` if there is no such value in the database. It will error if the index is negative or larger than the number of elements in the database.

See Also

`get_meta()`

get_origins *Get the origins of a value*

Description

`get_origins` returns the origins (package, function, parameter names) of a function given the hash of the value. It save a hash computation, compared to `get_origins_idx()`.

Usage

```
get_origins(db, hash)
```

Arguments

`db`	database, sxpdb object
`hash`	RAW, hash of the value to look for

Value

data frame with columns `pkg`, `fun` and `param` for package function and parameter names.

See Also

`get_origins_idx()`, `view_origins_db()`

---

get_origins_idx *Get the origins of a value*

Description

`get_origins` returns the origins (package, function, parameter names) of a function given the index of the value.

Usage

```
get_origins_idx(db, i)
```

Arguments

`db`	database, sxpdb object
`i`	integer, index of the value in the database

Value

data frame with columns `pkg`, `fun` and `param` for package function and parameter names.

See Also

`get_origins()`, `view_origins_db()`

---

get_value_idx *Get the value in the db at a given index*

Description

`get_value_idx` returns the value in `db` at index `idx`. You can also use the notation `db[idx]`.

Usage

```
get_value_idx(db, idx)
```

Arguments

`db`	database, sxpdb object
`idx`	integer, index in the database. Indexes start at 0 (not at 1!) so `idx >= 0` and `idx < size_db(db)`

Value

an R value

See Also

[`[.sxpdb()`]

`has_search_index` *Checks if the database has a search index*

Description

Requests to the database that pass a non-NULL query argument will not work if the search index is not built.

Usage

```
has_search_index(db)
```

Arguments

`db`	database, sxpdb object

Value

boolean, `TRUE` if it has a search index

See Also

`build_indexes()`

have_seen *Checks if a value is in the database*

Description

`have_seen` checks if a value is stored in the database and returns its index in that case.

Usage

```
have_seen(db, val)
```

Arguments

`db`	database, sxpdb object
`val`	R value

Value

integer index of the value if the value is in the db, `NULL` otherwise

is_query_empty *Checks if a query object is empty.*

Description

`is_query_empty` checks if the query refers to some values, or no values at all. You need to call it after at least It is typically used when the function using the query returns an ambiguous result (e.g. `sample_val()`) and you want to distinguish between `NULL` and an empty query that results also in `NULL`.

Usage

```
is_query_empty(query)
```

Arguments

`query`	query object

Value

boolean, whether the query is empty or not

See Also

`query_from_value()`, `query_from_plan()`, `relax_query()`, `close_query()`

map_db — *Map a function on the values in the database*

Description

`map_db` runs a function on each of the elements of the database matching a query and return a list of the results.

Usage

```
map_db(db, fun, query = NULL)
```

Arguments

`db`	database, sxpdb object
`fun`	R function, the function to apply to each value matching the query. It takes on argument, the value and should return an R value.
`query`	query object, typically built from `query_from_plan()` or `query_from_value()`.

See Also

`view_db()`, `filter_index_db()`, `view_meta_db()`, `view_origins_db()`

merge_all_dbs — *Merges several dbs into a new large one.*

Description

`merge_all_dbs` performs an efficient merge of several databases, to keep only unique values.

Usage

```
merge_all_dbs(db_paths, output_path, legacy = TRUE, parallel = TRUE)
```

Arguments

`db_paths`	character vector of paths to the databases to be merged together
`output_path`	character vector of the path of the resulting database
`legacy`	boolean In legacy mode, ”cran_db” is appended to the output_path
`parallel`	boolean, whether the merge should be performed in parallel or not.

Value

data frame with information about the merging process. The data frame has the following columns:

- `path`: path of the merged database
- `db_size_before`: size of the resulting database before merging the db at `path`
- `added_values`: number of new unique values added after merging the db at `path`
- `small_db_size`: number of values in the db at `path`
- `small_db_bytes`: size in bytes of the db at `path`
- `duration`: duration in seconds of merging the db at `path` into the resulting db
- `error`: whether there was an error when merging the db at `path`.

See Also

`merge_into()`

---

merge_db *Merge a db into another one.*

Description

Deprecated. Rather use `merge_into()`

Usage

```
merge_db(db1, db2)
```

Arguments

`db1`	database, sxpdb object
`db2`	database, sxpdb object

Value

`NULL` in case of error, number of values in the *target* database after merging otherwise

merge_into *Merge a source db into a target db*

Description

`merge_into` merges db *source* into db *target*. Db *target* must be open in write mode and db *source* must be opened in merge mode.

Usage

```
merge_into(target, source)
```

Arguments

`target`	database, sxpdb object
`source`	database, sxpdb object

Value

`NULL`in case of an error, a mapping from old indexes of all the values in the *source* database to the new indexes for them in the *target* database.

See Also

`merge_all_dbs()`, `build_indexes()`

nb_values_db *Computes the number of values matching a given query*

Description

`nb_values_db` returns the number of values matching a given query. This is very cheap to compute as it will just intersect or union some pre-computed arrays, so prefer to using `view_meta_db()` for simple quantitative questions. `nb_values_db` called with a `NULL` query is equivalent to calling `size_db`.

Usage

```
nb_values_db(db, query = NULL)
```

Arguments

`db`	database, sxpdb object
`query`	query object or `NULL`. If `NULL`, samples from the whole database

Value

integer, number of values matching the query.

See Also

`size_db()`

`open_db` *Open a database*

Description

`open_db` opens a database for reading, writing, or merging. Reading does not load as much in memory. For instance, it will directly read values from the disk. When merging database *source* into database *target*, *source* must be open in merge mode, and *target*, in write mode. Writing mode is required when adding any new values to the database or building or updating search indexes.

Usage

```
open_db(db = "db", mode = FALSE, quiet = TRUE)
```

Arguments

`db`	character vector of the name and path where to create the db. Default to `"db"`
`mode`	`TRUE` if in write mode, `FALSE` if in read mode (default), `"merge"` if in merge mode
`quiet`	boolean, whether to print messages or not

Value

`NULL` on error, a external pointer with class tag ”sxpdb” and class ”sxpdb” to the database on success

See Also

`close_db()`

path_db *Path to the database directory*

Description

The database is stored as a directory of various configuration files and tables. `path_db`returns the path to this directory.

Usage

```
path_db(db)
```

Arguments

`db`	database, sxpdb object

Value

character path to the database

query_from_plan *Creates a query from a plan.*

Description

`query_from_plan` creates a query from a plan describing the query. You can relax later on some parameters of the query with `relax_query()`. The query can then be used with any function taking a `query` argument.

Usage

```
query_from_plan(plan)
```

Arguments

`plan`	named list describing the query. In the absence of a parameter name, the associated metadata will be set to unspecified. The plan parameters are: • type: any value of the desired type • vector: boolean • length: integer • class name: character vector of class names, or boolean (to just specify that you want classes, but without specific class names) • na: boolean • ndims: integer • attributes: boolean • package: character vector of package names • func: character vector of function names

Value

query object

See Also

query_from_value(), relax_query(), close_query(), view_db(), map_db()

---

query_from_value *Creates a query from an example value.*

Description

query_from_value creates a query from an example R value. The query will mimic the value according to the following metadata: type, vector or not, length, class name, NA inside or not, number of dimensions, attributes or not. You can then relax on some of those metadata to state that you do not them want to be determined according to the example value, with relax_query(). The query can then be used with any function taking a query argument.

Usage

```
query_from_value(value)
```

Arguments

value	any R value

Value

query object

See Also

relax_query(), query_from_plan(), close_query(), view_db(), map_db()

---

relax_query *Relaxes on some parameters of a query.*

Description

relax_query makes a specified parameter from a query, i.e., with a given target value, into a non-specified parameter. For instance, if the query specifies that it wants values with length 34, relaxing on the length parameter will allow values with any lengths, not only values with lengths 34.

Usage

```
relax_query(query, relax)
```

Arguments

query	query object
relax	character vector none or several of "na", "length", "attributes", "type", "vector", "ndims", "class". You can also give "keep_type" or "keep_class" to relax on all constraints, except the type, or except the class names.

Value

boolean, TRUE if the query changed

See Also

query_from_value(), query_from_plan(), close_query(), is_query_empty()

---

sample_index *Sample randomly a value from the database*

Description

sample_index samples a value from the database and returns an index to the value. The sampling uses an uniform distribution. The value is picked along the whole database or according to a query built from query_from_plan() or query_from_value().

Usage

```
sample_index(db, query = NULL)
```

Arguments

db	database, sxpdb object
query	query object or NULL. If NULL, samples from the whole database

Value

integer, an index to a value in the database. You can then access the value with get_value_idx(). NULL if the query is empty.

See Also

sample_val()

sample_similar — *Sample randomly a value similar to a given from the database*

Description

`sample_similar` samples a value from the database, which is similar to `val` along metadata parameters, relaxed according to `relax`. The sampling uses an uniform distribution.

Usage

```
sample_similar(db, val, relax = "")
```

Arguments

`db`	database, sxpdb object
`val`	any R value
`relax`	character vector, as in `relax_query()`

Details

You should rather use a combination of `query_from_value()` and `sample_index()` for more control.

Value

an R value. If the query is empty, it will return `NULL`. It is ambiguous with sampling a value that happens to be `NULL` so you should rather use `sample_index()` with `query_from_value()`.

See Also

`sample_index()`, `query_from_value()`

sample_val — *Sample randomly a value from the database*

Description

`sample_val` samples a value from the database and returns the value. The sampling uses an uniform distribution. The value is picked along the whole database or according to a query built from `query_from_plan()` or `query_from_value()`.

Usage

```
sample_val(db, query = NULL)
```

Arguments

db	database, sxpdb object
query	query object or NULL. If NULL, samples from the whole database

Value

an R value. If the query is empty, it will return NULL. It is ambiguous with sampling a value that happens to be NULL so you should rather use sample_index().

See Also

sample_index()

---

show_query *Displays a query*

Description

show_query displays the content of a query, i.e. all its parameters. If a parameter is not defined, it shows any for its value. For class names, it shows a vector of class names, possibly an empty one.

Usage

```
show_query(query)
```

Arguments

query	query object

Value

integer, number of defined parameters

See Also

query_from_value(), query_from_plan(), relax_query(), close_query()

size_db *Number of elements in the database*

Description

`size_db` returns the number of elements in the database. All elements are unique.

Usage

```
size_db(db)
```

Arguments

`db`	database, sxpdb object

Value

integer, number of values in the database, or `NULL` in case of error

See Also

`nb_values_db()`

string_sexp_type *Get a textual repreentation of an R type*

Description

`string_sexp_type` shows the character representation of an R type encoded as an integer.

Usage

```
string_sexp_type(int_type)
```

Arguments

`int_type`	integer representation of a type

Value

character vector

values_from_calls *Fetches all the calls corresponding to one origin*

Description

`values_from_calls` fetches the values that share the given origin, i.e. that are one of the arguments or the return value of `pkg_name::fun_name`, and identify from which calls they come from.

Usage

```
values_from_calls(db, pkg_name, fun_name)
```

Arguments

`db`	database, sxpdb object
`pkg_name`	character vector of the name of the package
`fun_name`	character vector of the name of the function

Value

data frame with the following columns:

- `call_id`: unique id of the call
- `value_id`: unique id of the value
- `param`: parameter name

See Also

`values_from_origin()`, `view_origins_db()`, `view_call_ids()`

values_from_origin *Fetches all the values corresponding to one origin*

Description

`values_from_origins` fetches the values that share the given origin, i.e. that are one of the arguments or the return value of `pkg_name::fun_name`.

Usage

```
values_from_origin(db, pkg_name, fun_name)
```

Arguments

`db`	database, sxpdb object
`pkg_name`	character vector of the name of the package
`fun_name`	character vector of the name of the function

Value

data frame with two columns:

- `id`: ids of the values
- `param`: parameter names, concatenated and separated with `;`

See Also

`view_origins_db()`, `values_from_calls()`

`view_call_ids` *Fetches call ids from the database.*

Description

`view_call_ids` fetches call ids from the database, according to a given query.

Usage

```
view_call_ids(db, query = NULL)
```

Arguments

`db`	database, sxpdb object
`query`	not taken into account currently

Value

a data frame with column `call_id` where each cell is a list of integers, the call ids, and each row corresponds to a value. The values are ordered in the same way as their indexes in the database.

See Also

`map_db()` `get_meta_idx()` `view_db()` `map_db()` `filter_index_db()` `view_db_names()`

view_db — *Fetch values from the database.*

Description

`view_db` fetches values from the database, given a query. It will materialize the R values, and might quickly feel up memory. Rather use metadata (with `view_meta_db()`) if you don't need to look at the valeus itself. If you need to extract data from the values or transform them, rather use `map_db()`, which will only load one value at a time.

Usage

```
view_db(db, query = NULL)
```

Arguments

`db`	database, sxpdb object
`query`	query object, typically built from `query_from_plan()` or `query_from_value()`.

Value

list of values matching the query

See Also

`map_db()` `view_meta_db()` `filter_index_db()` `get_value_idx()`

view_db_names — *Fetches the origin databases*

Description

`view_db_names` fetches the origin databases of each values, i.e. in which database they were written first, if they result from merging in the current database.

Usage

```
view_db_names(db, query = NULL)
```

Arguments

`db`	database, sxpdb object
`query`	not taken into account currently

Value

a data frame with column `dbname` with the database names, in the same order as the values ordered by indexes in the database.

See Also

`map_db()` `get_meta_idx()` `view_db()` `map_db()` `filter_index_db()` `view_call_ids()`

`view_meta_db` *Fetches metadata from the database.*

Description

`view_meta_db` fetches metadata from the database, given a query. They include: * *runtime* metadata: number of calls the value has been seen in, number of times the value has been seen during merges * *static* metadata: type, size in bytes, length (for vector values), number of attributes, number of dimensionsm number of rows (for data frames, matrixes) * *debug* metadata: only if the database was created with sxpdb in debug mode, includes how many times `MAYBE_SHARED` has been true on the value, and how many times we were able to use the SEXP address optimization

Usage

```
view_meta_db(db, query = NULL)
```

Arguments

`db`	database, sxpdb object
`query`	query object, typically built from `query_from_plan()` or `query_from_value()`.

Value

data frame of the metadata for all the values, in the order of their indexes in the database

See Also

`map_db()` `get_meta_idx()` `view_db()` `map_db()` `filter_index_db()`

view_origins_db *Fetches origins from the database*

Description

`view_origins_db` fetches origins from the database, according to a given query.

Usage

```
view_origins_db(db, query = NULL)
```

Arguments

`db`	database, sxpdb object
`query`	query object, typically built from `query_from_plan()` or `query_from_value()`.

Value

data frame with columns `id`, `pkg`, `fun` and `param` for the index in the database, and package function and parameter names.

See Also

`get_origins()`, `get_origins_idx()`, `view_db()`, `map_db()`, `view_meta_db()`

write_mode *Checks if the database is in write mode*

Description

The database can be open in read mode, write mode or merge mode. `write_mode` checks if it is open in write mode.

Usage

```
write_mode(db)
```

Arguments

`db`	database, sxpdb object

Value

boolean, `TRUE` if it is in write mode

See Also

`open_db()`

Index