Package: harbouR (via r-universe)

November 3, 2026

Title Work with Self-Hosted 'SeaTable' Servers and Data Files

Version 0.1.0

Date 2026-10-19

Description An unofficial R client for self-hosted 'SeaTable' servers and for reading and writing 'SeaTable' data files directly. Reads a local '.dtable' export with no server at all, and talks to a self-hosted server over its REST (representational state transfer) application programming interface (API), returning tables, rows, columns and views as tidy 'tibble' objects through a column-type-aware coercion layer. Includes helpers for managing bases and files, converters to and from spreadsheet formats, and an optional 'shiny' explorer for interactive inspection. The 'SeaTable' API is documented at <https://api.seatable.com/>. This is an independent, third-party client: it is not affiliated with, endorsed by, or otherwise connected to SeaTable GmbH, and 'SeaTable' is their trademark, used here only to identify the software the package works with.

License MIT + file LICENSE

URL https://github.com/CTTIR/harbouR, https://cttir.github.io/harbouR/

BugReports https://github.com/CTTIR/harbouR/issues

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

Depends R (>= 4.1)

Imports cli, curl, httr2, jsonlite, lubridate, rlang, stats, tibble, tools, zip

Suggests bit64, bslib (>= 0.6.0), htmltools, knitr, reactable, readxl, rmarkdown, shiny, spelling, testthat (>= 3.2.0), tidyr, withr, writexl

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/Needs/website CTTIR/themakR, pkgdown

Config/Needs/coverage covr

Config/pak/sysreqs libssl-dev

Repository https://cttir.r-universe.dev

Date/Publication 2026-11-03 08:54:02 UTC

RemoteUrl https://github.com/CTTIR/harbouR

RemoteRef HEAD

RemoteSha 5aa7e76f3b968793fe41dd0b0fe7771bfc8e4ff9

Contents

as_tibble.harbour_dtable . . . 3
as_tibble.harbour_metadata . . . 4
harbouR-conditions . . . 5
hb_add_column . . . 7
hb_add_columns . . . 8
hb_add_select_option . . . 9
hb_append_rows . . . 10
hb_asset_path . . . 11
hb_attach_file . . . 11
hb_check_credentials . . . 12
hb_client . . . 13
hb_column_types . . . 15
hb_create_table . . . 16
hb_create_view . . . 17
hb_delete_asset . . . 18
hb_delete_column . . . 19
hb_delete_rows . . . 19
hb_delete_select_option . . . 20
hb_delete_table . . . 21
hb_delete_view . . . 22
hb_download_file . . . 23
hb_dtable . . . 24
hb_duplicate_table . . . 25
hb_example_metadata . . . 26
hb_example_rows . . . 27
hb_get_row . . . 28
hb_get_view . . . 29
hb_list_collaborators . . . 30
hb_list_columns . . . 30
hb_list_tables . . . 32
hb_list_views . . . 33
hb_lock_rows . . . 34

hb_metadata . . . 35
hb_ping . . . 35
hb_query . . . 36
hb_read_csv . . . 37
hb_read_dtable . . . 38
hb_read_table . . . 40
hb_read_xlsx . . . 41
hb_rename_table . . . 42
hb_run_explorer . . . 43
hb_server_info . . . 44
hb_unlock_rows . . . 45
hb_update_column . . . 46
hb_update_rows . . . 47
hb_update_select_option . . . 48
hb_update_view . . . 49
hb_upload_file . . . 50
hb_validate_dtable . . . 51
hb_write_csv . . . 52
hb_write_dtable . . . 53
hb_write_xlsx . . . 54
is_harbour_client . . . 55
is_harbour_dtable . . . 56
is_harbour_metadata . . . 57
length.harbour_dtable . . . 57
names.harbour_dtable . . . 58
print.harbour_dtable . . . 59
print.harbour_metadata . . . 59
summary.harbour_dtable . . . 60
summary.harbour_metadata . . . 61

Index **62**

`as_tibble.harbour_dtable` *Coerce a dtable's table index to a tibble*

Description

Coerce a dtable's table index to a tibble

Usage

```
## S3 method for class 'harbour_dtable'
as_tibble(x, ...)
```

Arguments

`x`	A `harbour_dtable`.
`...`	These dots are for future extensions and must be empty.

Value

A tibble with one row per table and columns `name` (chr), `n_rows` (int), `n_columns` (int) and `n_views` (int). Unlike the server path, the row count is real: the rows are in the file.

See Also

Other dtable: `hb_asset_path()`, `hb_dtable()`, `hb_read_csv()`, `hb_read_dtable()`, `hb_read_xlsx()`, `hb_validate_dtable()`, `hb_write_csv()`, `hb_write_dtable()`, `hb_write_xlsx()`, `is_harbour_dtable()`, `length.harbour_dtable()`, `names.harbour_dtable()`, `print.harbour_dtable()`, `summary.harbour_dtable()`

Examples

```
tibble::as_tibble(hb_dtable(Samples = data.frame(x = 1)))
```

---

as_tibble.harbour_metadata

Coerce harbour metadata to a tibble

Description

Coerce harbour metadata to a tibble

Usage

```
## S3 method for class 'harbour_metadata'
as_tibble(x, ...)
```

Arguments

`x`	A `harbour_metadata` object.
`...`	These dots are for future extensions and must be empty.

Value

A tibble with one row per table; see `hb_list_tables()`.

See Also

Other metadata: `hb_column_types()`, `hb_list_collaborators()`, `hb_list_tables()`, `hb_metadata()`, `is_harbour_metadata()`, `print.harbour_metadata()`, `summary.harbour_metadata()`

Examples

```
tibble::as_tibble(hb_example_metadata())
```

harbouR-conditions *Conditions signalled by harbouR*

Description

Every error harbouR raises itself, and every HTTP or transport failure, carries the class `harbour_error`, plus a more specific subclass, so callers can react to *what* went wrong rather than matching on message text. Errors from argument matching in rlang - a value that is not one of an argument's choices, or an argument passed in `...` that the function does not have - keep the classes rlang gives them.

Condition classes

`harbour_error_auth` Credentials are missing, malformed, or were rejected by the server. Fields: `status`, `url` and `response_body` when the server refused them; none when the problem was caught before any request. The token exchanges refuse credentials with whatever status their endpoint uses - a wrong password is HTTP 400, an unknown API token HTTP 403 - and such an error carries `harbour_error_http` or `harbour_error_permission` as well.

`harbour_error_http` The server returned a non-2xx status that has no more specific subclass, or the request got no complete answer (status `NA`). Fields: `status`, `url` and `response_body` - the parsed JSON, or the text when the server did not send JSON. It is also raised for a redirect (a 3xx status), which harbouR reports rather than follows, and for a reply that is not JSON whatever its status, a 2xx included; `response_body` then holds the start of its text. A request that got no answer - a refused connection, a DNS or TLS failure, a timeout, a dropped connection - also carries httr2's classes `httr2_failure` and `httr2_error`, and has no `response_body`. After a timeout or a dropped connection the server may already have applied a write, so a status of `NA` alone does not make sending the write again safe.

`harbour_error_not_found` HTTP 404, or a named table, view or column that does not exist in the base. Fields: `status`, `url`, `response_body` for the HTTP case.

`harbour_error_permission` HTTP 403. The token is not allowed to do this. Fields: `status`, `url`, `response_body`.

`harbour_error_rate_limit` HTTP 429. Fields: `status`, `url`, `response_body`.

`harbour_error_bad_argument` An argument failed validation before any request was made. Fields: none.

`harbour_error_unsupported` The operation is not available - for example an auth mode that this endpoint does not accept.

`harbour_error_column_collision` A column in the data would collide with a reserved SeaTable field name.

`harbour_error_package_asset` A packaged branding asset is missing from the installation. Field: `file`.

All of them also inherit from `harbour_error` and from `rlang_error`.

Warning classes

The warnings a script may want to check for, or to silence, carry a class of their own, and inherit from `rlang_warning`:

`harbour_warning_unknown_column` `hb_append_rows()` was given columns the table does not have, and drops them.

`harbour_warning_hidden_columns` `hb_read_table()` read through a view on a server, which leaves the columns the view hides out of its rows, so they are `NA`.

`harbour_warning_view_filters` `hb_read_table()` read through a filtered view of a `.dtable` file that records no rows for it, so every row was returned.

`harbour_warning_query_limit` `hb_query()` ran SQL without a `LIMIT` clause, so SeaTable returned at most 100 rows.

`harbour_warning_non_finite` `hb_append_rows()` or `hb_update_rows()` sent `Inf`, `-Inf` or `NaN`, which JSON cannot hold, as `null`. Field: `columns`.

`harbour_warning_checkbox_value` `hb_append_rows()` or `hb_update_rows()` sent a checkbox string that `as.logical()` cannot read as `TRUE` or `FALSE` as `null`. Field: `columns`.

`harbour_warning_asset_link` `hb_write_dtable()` left symbolic links in the asset tree, including a linked asset root, out of the archive.

`harbour_warning_dtable_tz` `hb_read_table()` read date-times from a `.dtable` that carry no time zone, as a server's export stores them, as UTC, because `options(harbouR.dtable_tz)` names none. Given once a session.

`harbour_warning_json_keys` `hb_write_dtable()` met object keys that JSON cannot keep - empty, or repeated - so those values change on the way to disk.

`harbour_warning_json_text` `hb_read_dtable()` met a NUL character in `content.json`, which R cannot keep in a string.

`harbour_warning_missing_assets` `hb_write_dtable()` could not include recorded attachments whose extracted files are missing. Field: `files`, their filenames inside the archive.

`harbour_warning_id_column` `hb_dtable()`, `hb_read_csv()` or `hb_read_xlsx()` dropped a column named `_id` that held values other than SeaTable row ids.

Examples

```
res <- tryCatch(
  hb_client(server = "https://example.org", api_token = NULL),
  harbour_error_auth = function(cnd) "no credentials"
)
res
```

hb_add_column *Add a column to a table*

Description

Add a column to a table

Usage

```
hb_add_column(client, table, name, type, ..., column_data = NULL)
```

Arguments

client	A harbour_client.
table	Table name.
name	Column name.
type	SeaTable column type, e.g. "text", "number", "date".
...	These dots are for future extensions and must be empty.
column_data	Optional list of column options, e.g. the choices for a select column. Named column_data rather than data because data means "the rows you are writing" everywhere else in harbouR.

Value

Invisibly returns the client.

See Also

Other columns: hb_add_columns(), hb_add_select_option(), hb_delete_column(), hb_delete_select_option(), hb_list_columns(), hb_update_column(), hb_update_select_option()

Examples

```
## Not run:
client <- hb_client()
hb_add_column(client, "Samples", "Notes", "text")

## End(Not run)
```

hb_add_columns *Add several columns at once*

Description

The columns are sent in one request. SeaTable's batch endpoint skips some column types without an error - a formula column among them - so any column its reply leaves out is then created on its own.

Usage

```
hb_add_columns(client, table, columns, ...)
```

Arguments

client	A harbour_client.
table	Table name.
columns	A list of column specs, each a named list with name, type and optionally data - the same shape hb_create_table() takes. SeaTable's own spelling, column_name, column_type and column_data, is accepted too.
...	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other columns: hb_add_column(), hb_add_select_option(), hb_delete_column(), hb_delete_select_option(), hb_list_columns(), hb_update_column(), hb_update_select_option()

Examples

```
## Not run:
client <- hb_client()
hb_add_columns(
  client, "Samples",
  list(
    list(name = "a", type = "text"),
    list(name = "b", type = "number")
  )
)

## End(Not run)
```

`hb_add_select_option` *Add a single-select option*

Description

The option is created in amber, with dark text: SeaTable requires both colours. Change them in SeaTable if you prefer others.

Usage

```
hb_add_select_option(client, table, name, option, ...)
```

Arguments

`client`	A `harbour_client`.
`table`	Table name.
`name`	Column name.
`option`	Option name to add.
`...`	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other columns: `hb_add_column()`, `hb_add_columns()`, `hb_delete_column()`, `hb_delete_select_option()`, `hb_list_columns()`, `hb_update_column()`, `hb_update_select_option()`

Examples

```
## Not run:
client <- hb_client()
hb_add_select_option(client, "Samples", "Status", "Done")

## End(Not run)
```

hb_append_rows *Append rows to a table*

Description

Append rows to a table

Usage

```
hb_append_rows(client, table, data, ..., chunk_size = 1000L)
```

Arguments

client	A harbour_client.
table	Table name.
data	A tibble or data frame whose columns match the table schema.
...	These dots are for future extensions and must be empty.
chunk_size	Rows per request. SeaTable caps batch writes at 1000 and harbouR clamps it, warning if you asked for more.

Value

Invisibly, a one-row tibble with columns table (chr), n_rows (int) - the count the server confirmed, or the number of rows sent when its reply gives none - and n_requests (int). The server's reply also lists the new rows' _ids; harbouR's summary does not include them.

See Also

Other rows: hb_delete_rows(), hb_get_row(), hb_lock_rows(), hb_query(), hb_read_table(), hb_unlock_rows(), hb_update_rows()

Examples

```
## Not run:
client <- hb_client()
hb_append_rows(client, "Samples", tibble::tibble(Name = "S1"))

## End(Not run)
```

hb_asset_path *Resolve a bundled asset URL to a local file*

Description

Attachment cells in an export point at `file://dtable-bundle/asset/...`. When the base was read with `assets = "extract"`, this gives the path the file was extracted to. The URL may be escaped, as SeaTable's export writes it - `my%20report%20%281%29.txt` for `my report (1).txt`.

Usage

```
hb_asset_path(x, url, ...)
```

Arguments

`x`	A `harbour_dtable` read with `assets = "extract"`.
`url`	An asset URL taken from a `file` or `image` cell.
`...`	These dots are for future extensions and must be empty.

Value

A single path, or `NA_character_` if the asset is not bundled.

See Also

Other dtable: `as_tibble.harbour_dtable()`, `hb_dtable()`, `hb_read_csv()`, `hb_read_dtable()`, `hb_read_xlsx()`, `hb_validate_dtable()`, `hb_write_csv()`, `hb_write_dtable()`, `hb_write_xlsx()`, `is_harbour_dtable()`, `length.harbour_dtable()`, `names.harbour_dtable()`, `print.harbour_dtable()`, `summary.harbour_dtable()`

Examples

```
path <- system.file("extdata", "example.dtable", package = "harbouR")
base <- hb_read_dtable(path, assets = "extract")
hb_asset_path(base, "file://dtable-bundle/asset/files/2026-07/readme.txt")
```

hb_attach_file *Attach a file to a cell*

Description

Convenience: uploads `path` then writes it into the given file or image column of the chosen row. A file column receives the file object; an image column, which holds URLs, receives the image's URL.

Usage

```
hb_attach_file(client, table, row_id, column, path, ...)
```

Arguments

client	A harbour_client.
table	Table name.
row_id	Row ID.
column	File or image column name.
path	Local file path. Must exist.
...	These dots are for future extensions and must be empty.

Details

The cell is **replaced**, not appended to: whatever it held before is no longer referenced by the row, although the earlier upload stays in the base's assets. To keep several attachments, read the cell, add the new object from hb_upload_file() and write it back with hb_update_rows().

Value

Invisibly returns the client.

See Also

Other files: hb_delete_asset(), hb_download_file(), hb_upload_file()

Examples

```
## Not run:
client <- hb_client()
hb_attach_file(client, "Samples", "abc", "Reports", "report.pdf")

## End(Not run)
```

---

hb_check_credentials *Check that the client's credentials are accepted*

Description

Exchanges the client's credentials for a token, which is the cheapest way to find out whether they work. For an API-token client this fetches a base token, and so also confirms the token is valid *for that base*; for a username/password client it fetches an account token and then drops any cached base token, so that the next base call fetches a new one.

Usage

```
hb_check_credentials(client, ...)
```

Arguments

client	A harbour_client.
...	These dots are for future extensions and must be empty.

Details

Unlike hb_ping(), which only proves the server is up, this proves the credentials are usable.

Value

The client, invisibly. Errors with a harbour_error_auth condition if the credentials are rejected.

See Also

hb_ping()

Other client: hb_client(), hb_ping(), hb_server_info(), is_harbour_client()

Examples

```
## Not run:
client <- hb_client()
hb_check_credentials(client)

## End(Not run)
```

---

hb_client *Create a SeaTable client*

Description

Builds a harbour_client object that holds the server URL, the user's credentials and a lazily-fetched, transparently-refreshed base token. The client is environment-backed, so internal token refreshes mutate cached state in place without reassigning your variable.

Usage

```
hb_client(
  server = Sys.getenv("SEATABLE_SERVER"),
  api_token = Sys.getenv("SEATABLE_API_TOKEN"),
  ...,
  username = NULL,
  password = NULL,
  base_uuid = NULL,
  workspace_id = NULL,
  base_name = NULL,
  timeout = 30
)
```

```
## S3 method for class 'harbour_client'
print(x, ...)

## S3 method for class 'harbour_client'
format(x, ...)
```

Arguments

server	URL of the SeaTable server, e.g. "https://seatable.example.org". Defaults to the SEATABLE_SERVER environment variable.
api_token	Long-lived per-base API token. Defaults to the SEATABLE_API_TOKEN env var, which is not read when username or password is given. Mutually exclusive with username/password.
...	These dots are for future extensions and must be empty.
username, password	Account credentials. Used to acquire an account token. Mutually exclusive with api_token.
base_uuid	Optional base UUID hint; usually discovered from the base-token exchange.
workspace_id, base_name	Which base to work on. Required for a username/password client, because an account token is scoped to the user rather than to a base: workspace_id is a positive whole number, or a string of its digits, and base_name a string. Ignored when api_token is used, since an API token names its base implicitly.
timeout	Per-request timeout in seconds, a positive number of at most 2147483, or Inf for no limit. Default 30. An upload or a download is not cut off after that long: it is limited to that long connecting, and a download to that long without moving data.
x	A harbour_client.

Details

SeaTable uses a three-token model. hb_client() accepts either an **API token** (per-base, long-lived, generated in the SeaTable UI) or **username + password** (which yields an account token). In-base calls transparently exchange these for a short-lived **base token**.

Defaults for server and api_token are read from the environment variables SEATABLE_SERVER and SEATABLE_API_TOKEN.

Value

A harbour_client object.

print() returns x, invisibly.

format() returns a character vector, one element per field.

See Also

Other client: hb_check_credentials(), hb_ping(), hb_server_info(), is_harbour_client()

Examples

```
# creating a client does not contact the server
client <- hb_client(
  server = "https://seatable.example.org",
  api_token = "your-api-token"
)
client

# a live base needs a SeaTable server and an API token
## Not run:
client <- hb_client(
  server = Sys.getenv("SEATABLE_SERVER"),
  api_token = Sys.getenv("SEATABLE_API_TOKEN")
)
print(client)

## End(Not run)
```

`hb_column_types` *SeaTable column types and how harbouR maps them*

Description

The single source of truth for the coercion layer: one row per type name, giving the R type harbouR produces when reading, whether that R type is a list-column, and whether harbouR leaves the column out of writes. The coercion functions and the column-types vignette are both derived from this table, so they cannot drift apart.

Usage

```
hb_column_types()
```

Details

`percent`, `dollar` and `euro` are number formats rather than column types: SeaTable reports such a column as type `"number"`, with the format in its `data$format`. They are listed so that tables carrying them as a type still coerce.

A `date` column reads as `POSIXct` in UTC. When the column's format has no time part, the cell holds a calendar date, and harbouR reads the date SeaTable shows as midnight UTC, whatever UTC offset the server sends with it, so `as.Date()` gives that date. A date-time read from a server is an instant and is converted to UTC; a date-time stored without an offset, as in a `.dtable` file, is read as UTC. A column that reports no format is read as a date-time.

Written to a date-only column, a `Date` is sent as itself and a date-time as the date it shows in its own time zone, except one at exactly midnight UTC, which keeps its UTC date: a cell read and written back keeps its day. A date-time column is sent the instant, in UTC.

Value

A tibble with one row per type and columns:

`seatable` chr. The type name as SeaTable reports it, or one of the number formats above.

`r` chr. The R type harbouR reads it as.

`is_list` lgl. Whether the result is a list-column.

`read_only` lgl. Whether harbouR leaves the column out of writes. Most of these are computed or maintained by SeaTable; a `digital-sign` cell is left out because a signature should come from the signer.

`notes` chr. Anything worth knowing.

See Also

Other metadata: `as_tibble.harbour_metadata()`, `hb_list_collaborators()`, `hb_list_tables()`, `hb_metadata()`, `is_harbour_metadata()`, `print.harbour_metadata()`, `summary.harbour_metadata()`

Examples

```
hb_column_types()

# the types harbouR does not write
subset(hb_column_types(), read_only)
```

---

hb_create_table *Create a table*

Description

Create a table

Usage

```
hb_create_table(client, table, ..., columns = list())
```

Arguments

`client`	A `harbour_client`.
`table`	Name of the new table.
`...`	These dots are for future extensions and must be empty.
`columns`	A list of column specifications: each element a named list with at least `name` and `type` (a SeaTable type string).

Value

Invisibly returns the client.

See Also

Other tables: hb_delete_table(), hb_duplicate_table(), hb_rename_table()

Examples

```
## Not run:
client <- hb_client()
hb_create_table(client, "NewTable",
  columns = list(list(name = "Name", type = "text")))

## End(Not run)
```

---

hb_create_view *Create a view*

Description

Create a view

Usage

```
hb_create_view(client, table, view, ..., settings = list())
```

Arguments

client	A harbour_client.
table	Table name.
view	New view name.
...	These dots are for future extensions and must be empty.
settings	Optional named list of further fields for the new view, such as type ("table", the default, or "archive") or is_locked. SeaTable sets filters, sorts and hidden columns only on an existing view, so follow with hb_update_view() for those.

Value

Invisibly returns the client.

See Also

Other views: hb_delete_view(), hb_get_view(), hb_list_views(), hb_update_view()

Examples

```
## Not run:
client <- hb_client()
hb_create_view(client, "Samples", "Active")

## End(Not run)
```

`hb_delete_asset` *Delete an asset*

Description

Removes a file or image from the base's assets. Like the other file endpoints this authenticates with the API token. Cells that still point at the asset are not changed.

Usage

```
hb_delete_asset(client, url, ...)
```

Arguments

`client`	A `harbour_client`.
`url`	Asset URL (as returned by SeaTable in a file/image cell).
`...`	These dots are for future extensions and must be empty.

Details

`url` is the URL from a file or image cell, absolute or relative to the server, or a path below the base's asset root such as `/files/2026-07/report.pdf`. The endpoint acts on the client's base whatever the URL names, so a URL on another server, or an asset URL of another base, raises a `harbour_error_bad_argument` condition rather than deleting this base's file at the same path.

Value

Invisibly returns the client.

See Also

Other files: `hb_attach_file()`, `hb_download_file()`, `hb_upload_file()`

Examples

```
## Not run:
client <- hb_client()
# A path below the asset root, or the URL from a file or image cell
hb_delete_asset(client, "/files/2026-07/report.pdf")

## End(Not run)
```

hb_delete_column *Delete a column*

Description

Delete a column

Usage

```
hb_delete_column(client, table, name, ...)
```

Arguments

client	A harbour_client.
table	Table name.
name	Column name.
...	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other columns: hb_add_column(), hb_add_columns(), hb_add_select_option(), hb_delete_select_option(), hb_list_columns(), hb_update_column(), hb_update_select_option()

Examples

```
## Not run:
client <- hb_client()
hb_delete_column(client, "Samples", "Old")

## End(Not run)
```

hb_delete_rows *Delete rows*

Description

Delete rows

Usage

```
hb_delete_rows(client, table, row_ids, ..., chunk_size = 1000L)
```

Arguments

client	A harbour_client.
table	Table name.
row_ids	A character vector of row IDs to delete.
...	These dots are for future extensions and must be empty.
chunk_size	Rows per request. SeaTable caps batch writes at 1000 and harbouR clamps it, warning if you asked for more.

Value

Invisibly, a one-row tibble with columns table (chr), n_rows (int) and n_requests (int).

See Also

Other rows: hb_append_rows(), hb_get_row(), hb_lock_rows(), hb_query(), hb_read_table(), hb_unlock_rows(), hb_update_rows()

Examples

```
## Not run:
client <- hb_client()
hb_delete_rows(client, "Samples", c("abc", "def"))

## End(Not run)
```

---

hb_delete_select_option

Delete a single-select option

Description

Delete a single-select option

Usage

```
hb_delete_select_option(client, table, name, option, ...)
```

Arguments

client	A harbour_client.
table	Table name.
name	Column name.
option	Name of the option to delete.
...	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other columns: hb_add_column(), hb_add_columns(), hb_add_select_option(), hb_delete_column(), hb_list_columns(), hb_update_column(), hb_update_select_option()

Examples

```
## Not run:
client <- hb_client()
hb_delete_select_option(client, "Samples", "Status", "Old")

## End(Not run)
```

---

hb_delete_table *Delete a table*

Description

Delete a table

Usage

```
hb_delete_table(client, table, ...)
```

Arguments

client	A harbour_client.
table	Name of the table to delete.
...	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other tables: hb_create_table(), hb_duplicate_table(), hb_rename_table()

Examples

```
## Not run:
client <- hb_client()
hb_delete_table(client, "DropMe")

## End(Not run)
```

`hb_delete_view` *Delete a view*

Description

SeaTable's API addresses one view by its name in the URL, and its server reads a slash in that name as a path separator and a name of one or two dots as a folder. Such a view cannot be deleted through the API, and harbouR says so before sending anything: rename it in SeaTable first.

Usage

```
hb_delete_view(client, table, view, ...)
```

Arguments

`client`	A `harbour_client`.
`table`	Table name.
`view`	View name.
`...`	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other views: `hb_create_view()`, `hb_get_view()`, `hb_list_views()`, `hb_update_view()`

Examples

```
## Not run:
client <- hb_client()
hb_delete_view(client, "Samples", "Old")

## End(Not run)
```

`hb_download_file` *Download an asset*

Description

A URL in a file or image cell is not directly downloadable: the asset sits behind SeaTable's login. harbouR recognises such a URL by its path, `/workspace/<id>/asset/<base uuid>/...`, whether it is absolute or relative to the server, and exchanges it, with the client's API token, for a short-lived download link. The URL must name the client's server - the host is compared ignoring case, the scheme and a default port - and the client's base: an asset of another server or another base raises a `harbour_error_bad_argument` condition instead of being fetched. A URL that is not an asset URL is fetched as it is.

Usage

```
hb_download_file(client, url, dest, ..., overwrite = FALSE)
```

Arguments

`client`	A `harbour_client`.
`url`	Asset URL (as returned by SeaTable in a file/image cell).
`dest`	Destination path. Parent directories are created if needed.
`...`	These dots are for future extensions and must be empty.
`overwrite`	Refuse to clobber an existing file unless `TRUE`.

Details

Should SeaTable answer an asset download with its login page, the download fails with a `harbour_error_http` condition rather than saving the page in place of the file.

The file is written to a temporary name beside `dest` and moved into place only once complete, so a failed download never leaves a partial file and never destroys the file `overwrite = TRUE` would replace.

Value

Invisibly returns `dest`.

See Also

Other files: `hb_attach_file()`, `hb_delete_asset()`, `hb_upload_file()`

Examples

```
## Not run:
client <- hb_client()
hb_download_file(client, "https://...", tempfile())

## End(Not run)
```

`hb_dtable` *Build a* `.dtable` *base from data frames*

Description

Creates a `harbour_dtable` from scratch, so a set of R data frames can be written out as a SeaTable base and imported.

Usage

```
hb_dtable(..., base_name = "harbouR base", collaborators = NULL)
```

Arguments

`...`	Named data frames, one per table.
`base_name`	Name recorded for the base.
`collaborators`	Optional list of collaborator records.

Details

Column types are inferred: `character` becomes `text`, `numeric` `number`, `logical` `checkbox`, `Date` and `POSIXct` `date`, `factor` `single-select` with its levels as options, and a list-column of character vectors `multiple-select` with the values it uses as options. A list-column holding objects - file, image or location cells read from a live base - is flattened to `text`: a new base has none of the assets those cells point at.

A `Date` column, or a date-time column whose every value is midnight UTC - which is how `hb_read_table()` reads a date-only column - becomes a date column showing the date alone. Other date-times are written with their UTC offset, so the server stores the instant whatever zone it runs in.

A column named `_id`, such as the one `hb_read_table()` adds, is dropped: SeaTable keeps that name for the row identifier, and the new base has identifiers of its own. One that holds anything but SeaTable row ids is dropped with a `harbour_warning_id_column` warning.

New row ids, table ids, column keys and select-option ids are drawn from the session's random number generator, so building a base advances `.Random.seed`: a random draw made after `hb_dtable()` differs from the same draw made without it. Set the seed after building the base when an analysis needs its random numbers not to depend on it.

Value

A `harbour_dtable`, with the components `hb_read_dtable()` describes, holding one table per data frame.

See Also

Other dtable: `as_tibble.harbour_dtable()`, `hb_asset_path()`, `hb_read_csv()`, `hb_read_dtable()`, `hb_read_xlsx()`, `hb_validate_dtable()`, `hb_write_csv()`, `hb_write_dtable()`, `hb_write_xlsx()`, `is_harbour_dtable()`, `length.harbour_dtable()`, `names.harbour_dtable()`, `print.harbour_dtable()`, `summary.harbour_dtable()`

Examples

```
base <- hb_dtable(
  Samples = data.frame(Name = c("a", "b"), Value = c(1.5, 2.5)),
  base_name = "My base"
)
base

hb_read_table(base, "Samples")
```

`hb_duplicate_table` *Duplicate a table*

Description

Duplicate a table

Usage

```
hb_duplicate_table(client, table, ..., duplicate_records = TRUE)
```

Arguments

`client`	A `harbour_client`.
`table`	Source table name.
`...`	These dots are for future extensions and must be empty.
`duplicate_records`	Copy the rows as well as the structure. Default `TRUE`.

Value

Invisibly returns the client.

Naming

SeaTable names the copy after the original with `(copy)` appended, with no space - a copy of `Samples` is `Samples(copy)` - and offers no way to set the name in the same request. Follow with `hb_rename_table()` if you need a particular name.

See Also

Other tables: `hb_create_table()`, `hb_delete_table()`, `hb_rename_table()`

Examples

```
## Not run:
client <- hb_client()
hb_duplicate_table(client, "Samples")
hb_rename_table(client, "Samples(copy)", "Samples_backup")

## End(Not run)
```

`hb_example_metadata` *Offline example metadata*

Description

A small, hand-crafted SeaTable base used to power offline examples and tests. The shape of the returned object matches what the SeaTable metadata endpoint would yield.

Usage

```
hb_example_metadata()
```

Details

The base is defined once, in `inst/extdata/example_metadata.json`, and parsed on each call. Keeping it in a data file rather than in R source means the demo base is also readable from outside R, and there is only one definition to keep in step with the coercion layer.

Value

A `harbour_metadata` object describing a tiny base with two tables: `Samples` and `Patients`.

See Also

Other example data: `hb_example_rows()`

Examples

```
meta <- hb_example_metadata()
tibble::as_tibble(meta)
```

hb_example_rows *Offline example rows*

Description

Returns a tibble of example rows for one of the tables provided by hb_example_metadata(), so tests and the column-types vignette can demonstrate the coercion layer without a SeaTable server.

Usage

```
hb_example_rows(table = c("Samples", "Patients"))
```

Arguments

table	One of "Samples" or "Patients".

Details

Between them the two tables cover the common column types - text, number, single-select, multiple-select, date, collaborator, file and checkbox - so the rows include list-columns and date-times. For the fuller set, read the bundled .dtable file with hb_read_dtable(): it carries a column of nearly every type, at system.file("extdata", "example.dtable", package = "harbouR").

Value

A tibble with one row per example record: a column for each column of table in hb_example_metadata(), and the row id _id. Multiple-select, collaborator and file columns are list-columns, and dates are date-times in UTC.

See Also

Other example data: hb_example_metadata()

Examples

```
hb_example_rows("Samples")
```

hb_get_row *Get a single row by ID*

Description

Get a single row by ID

Usage

```
hb_get_row(client, table, row_id, ...)
```

Arguments

client	A harbour_client.
table	Table name.
row_id	The SeaTable row identifier.
...	These dots are for future extensions and must be empty.

Value

A one-row tibble with one column per SeaTable column, plus _id (chr). An unknown row_id is an error - a harbour_error_not_found condition - not an empty result, because asking for a specific row that does not exist is a mistake worth surfacing.

See Also

Other rows: hb_append_rows(), hb_delete_rows(), hb_lock_rows(), hb_query(), hb_read_table(), hb_unlock_rows(), hb_update_rows()

Examples

```
## Not run:
client <- hb_client()
hb_get_row(client, "Samples", "abc123")

## End(Not run)
```

hb_get_view *Get a view's settings*

Description

SeaTable's API addresses one view by its name in the URL, and its server reads a slash in that name as a path separator and a name of one or two dots as a folder. Such a view is looked up in the table's list of views instead.

Usage

```
hb_get_view(client, table, view, ...)
```

Arguments

client	A harbour_client.
table	Table name.
view	View name.
...	These dots are for future extensions and must be empty.

Value

A one-row tibble with columns name (chr), type (chr), is_default (lgl), filters (list), sorts (list) and hidden_columns (list of chr).

See Also

Other views: hb_create_view(), hb_delete_view(), hb_list_views(), hb_update_view()

Examples

```
## Not run:
client <- hb_client()
hb_get_view(client, "Samples", "Default")

## End(Not run)
```

hb_list_collaborators *List collaborators of the active base*

Description

List collaborators of the active base

Usage

```
hb_list_collaborators(client, ...)
```

Arguments

client	A harbour_client.
...	These dots are for future extensions and must be empty.

Value

A tibble with columns email (chr), name (chr) and contact_email (chr). Zero rows if none are reported.

See Also

Other metadata: as_tibble.harbour_metadata(), hb_column_types(), hb_list_tables(), hb_metadata(), is_harbour_metadata(), print.harbour_metadata(), summary.harbour_metadata()

Examples

```
## Not run:
client <- hb_client()
hb_list_collaborators(client)

## End(Not run)
```

hb_list_columns *List columns of a table*

Description

List columns of a table

Usage

```
hb_list_columns(x, table, ...)

## S3 method for class 'harbour_client'
hb_list_columns(x, table, ..., refresh = FALSE)

## S3 method for class 'harbour_dtable'
hb_list_columns(x, table, ...)

## Default S3 method:
hb_list_columns(x, table, ...)
```

Arguments

x	A harbour_client connected to a base, or a harbour_dtable read from a local file.
table	Table name.
...	These dots are for future extensions and must be empty.
refresh	Logical. Ask the server rather than reusing the cached base metadata. Default FALSE.

Value

A tibble with one row per column and columns name (chr), type (chr), key (chr), editable (lgl) and data (list). editable says whether harbouR writes the column: it is FALSE for the types hb_column_types() marks read_only. SeaTable's own editable flag is not used, since it is set for formula columns too. The data list-column holds the column's type-specific configuration - for a select column, its options.

See Also

Other columns: hb_add_column(), hb_add_columns(), hb_add_select_option(), hb_delete_column(), hb_delete_select_option(), hb_update_column(), hb_update_select_option()

Examples

```
# a .dtable file needs no server
path <- system.file("extdata", "example.dtable", package = "harbouR")
hb_list_columns(hb_read_dtable(path), "Samples")

# a live base needs a SeaTable server and an API token
## Not run:
client <- hb_client()
hb_list_columns(client, "Samples")

## End(Not run)
```

`hb_list_tables` *List the tables in a base*

Description

List the tables in a base

Usage

```
hb_list_tables(x, ...)

## S3 method for class 'harbour_dtable'
hb_list_tables(x, ...)

## Default S3 method:
hb_list_tables(x, ...)

## S3 method for class 'harbour_client'
hb_list_tables(x, ..., refresh = FALSE)
```

Arguments

`x`	A `harbour_client` connected to a base, or a `harbour_dtable` read from a local file.
`...`	These dots are for future extensions and must be empty.
`refresh`	Logical; refetch metadata even if cached. Default `FALSE`.

Value

A tibble with one row per table and columns `name` (chr), `n_columns` (int) and `n_views` (int).

The server path stops there: the metadata endpoint carries no row payloads, so there is no row count to report - use `hb_read_table()` if you need one. The `.dtable` path additionally returns `n_rows` (int), because in a file the rows are right there.

See Also

Other metadata: `as_tibble.harbour_metadata()`, `hb_column_types()`, `hb_list_collaborators()`, `hb_metadata()`, `is_harbour_metadata()`, `print.harbour_metadata()`, `summary.harbour_metadata()`

Examples

```
# a .dtable file needs no server
path <- system.file("extdata", "example.dtable", package = "harbouR")
hb_list_tables(hb_read_dtable(path))

# a live base needs a SeaTable server and an API token
## Not run:
```

```
client <- hb_client()
hb_list_tables(client)

## End(Not run)
```

`hb_list_views` *List views of a table*

Description

List views of a table

Usage

```
hb_list_views(x, table, ...)

## S3 method for class 'harbour_dtable'
hb_list_views(x, table, ...)

## Default S3 method:
hb_list_views(x, table, ...)

## S3 method for class 'harbour_client'
hb_list_views(x, table, ..., refresh = FALSE)
```

Arguments

`x`	A `harbour_client` connected to a base, or a `harbour_dtable` read from a local file.
`table`	Table name.
`...`	These dots are for future extensions and must be empty.
`refresh`	Logical. Ask the server rather than reusing the cached base metadata. Default `FALSE`.

Value

A tibble with columns `name` (chr), `type` (chr) and `is_default` (lgl). Zero rows if no views exist.

See Also

Other views: `hb_create_view()`, `hb_delete_view()`, `hb_get_view()`, `hb_update_view()`

Examples

```
# a .dtable file needs no server
path <- system.file("extdata", "example.dtable", package = "harbouR")
hb_list_views(hb_read_dtable(path), "Samples")

# a live base needs a SeaTable server and an API token
## Not run:
client <- hb_client()
hb_list_views(client, "Samples")

## End(Not run)
```

`hb_lock_rows` *Lock rows*

Description

Lock rows

Usage

```
hb_lock_rows(client, table, row_ids, ...)
```

Arguments

`client`	A `harbour_client`.
`table`	Table name.
`row_ids`	A character vector of row IDs to lock.
`...`	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other rows: `hb_append_rows()`, `hb_delete_rows()`, `hb_get_row()`, `hb_query()`, `hb_read_table()`, `hb_unlock_rows()`, `hb_update_rows()`

Examples

```
## Not run:
client <- hb_client()
hb_lock_rows(client, "Samples", "abc")

## End(Not run)
```

hb_metadata *Fetch base metadata*

Description

Retrieves the structural metadata for a SeaTable base - the list of tables, their columns (with SeaTable types) and views. The result is cached on the client.

Usage

```
hb_metadata(client, ...)
```

Arguments

client	A harbour_client.
...	These dots are for future extensions and must be empty.

Value

A harbour_metadata object - a list with components base_name (chr), tables (list), version (chr).

See Also

Other metadata: as_tibble.harbour_metadata(), hb_column_types(), hb_list_collaborators(), hb_list_tables(), is_harbour_metadata(), print.harbour_metadata(), summary.harbour_metadata()

Examples

```
## Not run:
client <- hb_client()
meta <- hb_metadata(client)
tibble::as_tibble(meta)

## End(Not run)
```

hb_ping *Check that a SeaTable server is reachable*

Description

Issues a lightweight, unauthenticated request against the server's ping endpoint. This tests *connectivity only* - use hb_check_credentials() to test whether the client's credentials are accepted.

Usage

```
hb_ping(client, ...)
```

Arguments

client	A harbour_client.
...	These dots are for future extensions and must be empty.

Details

Returns the client invisibly, so it composes: client |> hb_ping() |> hb_read_table("Samples").

Value

The client, invisibly. Errors with a harbour_error_http condition if the server cannot be reached, and with the class of the answer - harbour_error_not_found for a 404, say - if it answers with an error.

See Also

hb_check_credentials()

Other client: hb_check_credentials(), hb_client(), hb_server_info(), is_harbour_client()

Examples

```
## Not run:
client <- hb_client()
hb_ping(client)

## End(Not run)
```

---

hb_query *Run a SeaTable SQL query*

Description

Run a SeaTable SQL query

Usage

```
hb_query(client, sql, ..., parameters = NULL, convert_keys = TRUE)
```

Arguments

client	A harbour_client.
sql	SeaTable SQL query string. SeaTable applies an implicit LIMIT 100 when the query has none, and caps results at 10000 rows. harbouR warns if there is no LIMIT clause - the word inside a quoted string or a back-ticked name does not count - at most once every eight hours for the same query, so a loop does not repeat it.
...	These dots are for future extensions and must be empty.

parameters	Optional list of values for ? placeholders in sql, in order. Names are ignored.
convert_keys	Return column names rather than column keys. Default TRUE.

Value

A tibble, typed from the result schema SeaTable reports alongside the rows, so the same table read via hb_read_table() and via hb_query() yields the same column types. Always a tibble, even when the query returns no rows.

See Also

Other rows: hb_append_rows(), hb_delete_rows(), hb_get_row(), hb_lock_rows(), hb_read_table(), hb_unlock_rows(), hb_update_rows()

Examples

```
## Not run:
client <- hb_client()
hb_query(client, "select * from Samples limit 5")

## End(Not run)
```

---

hb_read_csv *Read tables into a new base from CSV files*

Description

Builds a fresh harbour_dtable from CSV files, one table per file. This is **not** the inverse of hb_write_csv(): it produces a new base with new identifiers, and only text, number and checkbox columns. Nothing that a CSV cannot carry is recovered.

Usage

```
hb_read_csv(files, ..., base_name = "harbouR base")
```

Arguments

files	Paths to .csv files. Table names come from the file names unless files is named.
...	These dots are for future extensions and must be empty.
base_name	Name recorded for the new base.

Details

Columns are typed the way `utils::read.csv()` types them, and quoting a value does not change that:

- A column of numbers becomes `number`, and one holding only `T`, `F`, `TRUE` or `FALSE` becomes `checkbox`.
- A value written `NA` is read as missing.
- Dates and date-times arrive as text.

Some exceptions avoid losing data. A column holding an integer with a leading zero, such as `"0001"`, a value with a leading plus, such as the telephone number `"+4917612345678"`, or an integer of more than 15 digits stays text, rather than losing the zeros, the plus or digits, and so does one holding a code such as `"12E"`, which `utils::read.csv()` reads as 12 in R before 4.4.1. A column with no values at all is text, not a checkbox column that would read `FALSE` in every row.

The files must be UTF-8, with or without a byte-order mark. A file in another encoding is refused, naming the first line that is not valid UTF-8. In a multibyte session that is not UTF-8, such as Shift-JIS or GBK, the text goes through that session's encoding, so characters it lacks are lost; read such files in a UTF-8 session. A column named `_id`, such as the one `hb_write_csv()` writes, is dropped: SeaTable keeps that name for the row identifier, and the new base has identifiers of its own. One that holds anything but SeaTable row ids, such as a database's keys, is dropped with a warning.

Value

A `harbour_dtable`, with the components `hb_read_dtable()` describes, holding one table per file.

See Also

Other dtable: `as_tibble.harbour_dtable()`, `hb_asset_path()`, `hb_dtable()`, `hb_read_dtable()`, `hb_read_xlsx()`, `hb_validate_dtable()`, `hb_write_csv()`, `hb_write_dtable()`, `hb_write_xlsx()`, `is_harbour_dtable()`, `length.harbour_dtable()`, `names.harbour_dtable()`, `print.harbour_dtable()`, `summary.harbour_dtable()`

Examples

```
csv <- tempfile(fileext = ".csv")
utils::write.csv(data.frame(x = 1:2, y = c("a", "b")), csv,
                 row.names = FALSE)
hb_read_csv(c(Measurements = csv))
```

---

hb_read_dtable *Read a SeaTable* `.dtable` *file*

Description

Opens a `.dtable` export and returns a `harbour_dtable` object. The same verbs that work against a live base work against it - `hb_list_tables()`, `hb_read_table()`, `hb_list_columns()` - so an analysis can be written once and run either way.

Usage

```
hb_read_dtable(path, ..., assets = c("none", "extract"), assets_dir = NULL)
```

Arguments

path	Path to a .dtable file.
...	These dots are for future extensions and must be empty.
assets	Whether to extract the bundled asset/ tree. "none", the default, reads only content.json. "extract" unpacks the assets so hb_asset_path() can resolve attachment URLs to local files. An archive whose asset entries could land outside assets_dir - a name that is absolute or has a .. component, between slashes or backslashes, or an entry that is a symbolic link - is refused before anything is extracted. Assets whose names differ only in case may not all be extracted, and a warning names them. Existing paths inside assets_dir are checked before extraction: links leading outside it or whose targets are missing are refused.
assets_dir	Where to extract assets to. Defaults to a session temporary directory. A folder that already holds an asset/ tree is refused: the next hb_write_dtable() would pack that tree as well.

Details

A .dtable is a ZIP archive containing content.json, the complete base, and optionally an asset/ tree of uploaded files and images.

The parsed JSON is kept **verbatim**. Real exports carry fields no client would think to model - one column in the reference file holds a serialised React element - and future SeaTable releases will add more. Keeping the tree untouched is what makes hb_write_dtable() lossless; rebuilding it from a typed intermediate would quietly drop whatever harbouR did not know about.

Value

A harbour_dtable: a list with components content (the parsed tree), path, assets (the archive's own listing of its asset/ tree, as zip::zip_list() gives it: one row per entry, folders included), assets_dir and base_name. Note that names() on a harbour_dtable gives its *table* names, not these components - reach them with $ or unclass().

Options

- harbouR.dtable_tz: the time zone of the SeaTable server a file comes from, such as "Europe/Berlin". A server stores a date-time as the time it shows, with no offset, and its export keeps it so; such a value is read in this zone. Unset, it is read as UTC, with a warning once a session. Date-only cells, and values that carry an offset, need no zone.
- harbouR.max_dtable_bytes: how large content.json, plus the assets when they are extracted, may be once unpacked. Default 4 GiB. A larger file is refused before anything is read, since a small archive can unpack to enough to fill memory or disk.

See Also

hb_write_dtable(), hb_dtable()

Other dtable: as_tibble.harbour_dtable(), hb_asset_path(), hb_dtable(), hb_read_csv(), hb_read_xlsx(), hb_validate_dtable(), hb_write_csv(), hb_write_dtable(), hb_write_xlsx(), is_harbour_dtable(), length.harbour_dtable(), names.harbour_dtable(), print.harbour_dtable(), summary.harbour_dtable()

Examples

```
path <- system.file("extdata", "example.dtable", package = "harbouR")
base <- hb_read_dtable(path)
base

hb_list_tables(base)
hb_read_table(base, "Samples")
```

---

hb_read_table *Read a table as a tibble*

Description

Reads all rows from `table` (optionally filtered by `view`) and returns them as a typed tibble. Pagination is handled internally; the returned tibble always has the table's columns in declared order plus an `_id` column.

Usage

```
hb_read_table(x, table, ...)

## S3 method for class 'harbour_dtable'
hb_read_table(x, table, ..., view = NULL, n_max = Inf, option_labels = TRUE)

## Default S3 method:
hb_read_table(x, table, ...)

## S3 method for class 'harbour_client'
hb_read_table(x, table, ..., view = NULL, page_size = 1000L, n_max = Inf)
```

Arguments

`x`	A `harbour_client` connected to a base, or a `harbour_dtable` read from a local file.
`table`	Name of the table.
`...`	These dots are for future extensions and must be empty.
`view`	Optional view name. A server returns only the columns a view shows, so a read through a view gives `NA` for the columns it hides, and harbouR warns naming them. A `.dtable` read keeps their values.

n_max	Maximum number of rows to return. Inf, the default, reads the whole table.
option_labels	Translate select-option ids to their display names. On disk a select cell holds an option id; over the API it holds the name, so this is what makes a local read and a server read of the same column agree. Set FALSE to see the raw ids.
page_size	Rows fetched per request. SeaTable caps this at 1000 and harbouR clamps it, warning if you asked for more.

Value

A tibble with one row per SeaTable row and one column per SeaTable column, plus _id (chr). A 0-row tibble is returned for an empty table.

Views on a .dtable file

On a harbour_dtable, view restricts the rows only to the row ids the file records for that view, kept in table order. The view's filters and sorts are not applied. A view that records no rows returns the whole table, with a warning of class harbour_warning_view_filters when the view has filters that were therefore not applied.

See Also

Other rows: hb_append_rows(), hb_delete_rows(), hb_get_row(), hb_lock_rows(), hb_query(), hb_unlock_rows(), hb_update_rows()

Examples

```
# a .dtable file needs no server
path <- system.file("extdata", "example.dtable", package = "harbouR")
base <- hb_read_dtable(path)
hb_read_table(base, "Samples")
hb_read_table(base, "Samples", n_max = 1)

# a live base needs a SeaTable server and an API token
## Not run:
client <- hb_client()
hb_read_table(client, "Samples")

# just the first 10 rows
hb_read_table(client, "Samples", n_max = 10)

## End(Not run)
```

---

hb_read_xlsx *Read tables into a new base from an Excel workbook*

Description

One table per worksheet. Like hb_read_csv(), this produces a *new* base rather than restoring one.

Usage

```
hb_read_xlsx(path, ..., sheets = NULL, base_name = "harbouR base")
```

Arguments

path	Path to an .xlsx file.
...	These dots are for future extensions and must be empty.
sheets	Worksheet names to read. Default: all of them.
base_name	Name recorded for the new base.

Details

A column's type follows its cells, judged over every row of the sheet. A column with no values at all is text. The spaces around a text value are kept. A column named _id, such as the one hb_write_xlsx() writes, is dropped, with a warning when it holds anything but SeaTable row ids.

Value

A harbour_dtable, with the components hb_read_dtable() describes, holding one table per sheet read.

See Also

Other dtable: as_tibble.harbour_dtable(), hb_asset_path(), hb_dtable(), hb_read_csv(), hb_read_dtable(), hb_validate_dtable(), hb_write_csv(), hb_write_dtable(), hb_write_xlsx(), is_harbour_dtable(), length.harbour_dtable(), names.harbour_dtable(), print.harbour_dtable(), summary.harbour_dtable()

Examples

```
path <- tempfile(fileext = ".xlsx")
writexl::write_xlsx(list(Samples = data.frame(x = 1:2)), path)
hb_read_xlsx(path)
```

---

hb_rename_table *Rename a table*

Description

Rename a table

Usage

```
hb_rename_table(client, table, new_name, ...)
```

Arguments

client	A harbour_client.
table	Current table name.
new_name	New table name.
...	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other tables: hb_create_table(), hb_delete_table(), hb_duplicate_table()

Examples

```
## Not run:
client <- hb_client()
hb_rename_table(client, "Old", "New")

## End(Not run)
```

---

hb_run_explorer *Launch the harbouR explorer*

Description

A Shiny app for working with a SeaTable base without writing code: open a local .dtable export or connect to a server, browse the tables, read the schema, run SQL, and download the result as .dtable, Excel or CSV.

Usage

```
hb_run_explorer(x = NULL, ..., host = "127.0.0.1", port = NULL)
```

Arguments

x	Optional harbour_client or harbour_dtable to open with.
...	Passed to shiny::runApp().
host	Host to bind. Default "127.0.0.1", this computer only. The app has no login: bound to an address others can reach, it hands them a client passed to it - its token, and a SQL box that can change the base - and whatever file is open.
port	Port. Default NULL, letting Shiny choose.

Details

Pass a connected `hb_client()` or a base read with `hb_read_dtable()` to start there. With no argument the app opens on its source panel, which offers a bundled example base that needs no credentials and no network.

The UI packages (`shiny`, `bslib`, `reactable`) are in `Suggests`, so the client itself stays headless. A missing one, or a `bslib` older than 0.6.0, produces a single informative error rather than a stack trace. The Excel download also needs `writexl`; without it the app says so when you ask for one.

Shiny refuses uploads over 5 MB by default, and a `.dtable` with attachments is often larger, so while the app runs it accepts files up to 1 GB. To choose another limit, set the `shiny.maxRequestSize` option, in bytes, before calling `hb_run_explorer()`; a value you have set is left as it is.

Value

Invisible `NULL`; launches a Shiny application.

Examples

```
# the bundled example base, no credentials needed
hb_run_explorer()

# or start from a file, with its attachments, so that a .dtable
# download keeps them
path <- system.file("extdata", "example.dtable", package = "harbouR")
hb_run_explorer(hb_read_dtable(path, assets = "extract"))
```

`hb_server_info` *Server information*

Description

Returns the SeaTable server's reported version and basic info. This endpoint is unauthenticated.

Usage

```
hb_server_info(client, ...)
```

Arguments

`client`	A `harbour_client`.
`...`	These dots are for future extensions and must be empty.

Value

A one-row tibble with columns `server` (chr), `version` (chr) and `edition` (chr) where reported.

See Also

Other client: hb_check_credentials(), hb_client(), hb_ping(), is_harbour_client()

Examples

```
## Not run:
client <- hb_client()
hb_server_info(client)

## End(Not run)
```

---

hb_unlock_rows *Unlock rows*

Description

Unlock rows

Usage

```
hb_unlock_rows(client, table, row_ids, ...)
```

Arguments

client	A harbour_client.
table	Table name.
row_ids	A character vector of row IDs to unlock.
...	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other rows: hb_append_rows(), hb_delete_rows(), hb_get_row(), hb_lock_rows(), hb_query(), hb_read_table(), hb_update_rows()

Examples

```
## Not run:
client <- hb_client()
hb_unlock_rows(client, "Samples", "abc")

## End(Not run)
```

hb_update_column *Update a column*

Description

Update a column

Usage

```
hb_update_column(
  client,
  table,
  name,
  ...,
  new_name = NULL,
  new_type = NULL,
  column_data = NULL
)
```

Arguments

client	A harbour_client.
table	Table name.
name	Column name.
...	These dots are for future extensions and must be empty.
new_name	Optional new column name. Sends op_type = "rename_column".
new_type	Optional new SeaTable column type. Sends op_type = "modify_column_type". SeaTable performs one operation per request, so this cannot be combined with new_name.
column_data	Optional list of column options, sent with the type change. Given without new_type, it reconfigures the column under its current type, which harbouR asks the server for - SeaTable requires the type to be named.

Value

Invisibly returns the client.

See Also

Other columns: hb_add_column(), hb_add_columns(), hb_add_select_option(), hb_delete_column(), hb_delete_select_option(), hb_list_columns(), hb_update_select_option()

Examples

```
## Not run:
client <- hb_client()
hb_update_column(client, "Samples", "Notes", new_name = "Comments")

## End(Not run)
```

`hb_update_rows` *Update rows in a table*

Description

Every column of `data` that the table can store is written, so an `NA` clears its cell. The exception is a tibble read through a view with `hb_read_table()`: the columns the view hides come back as `NA`, and a cell of theirs is left out of the write unless it has been filled in, so writing such a tibble back does not erase them.

Usage

```
hb_update_rows(
  client,
  table,
  data,
  ...,
  row_id_col = "_id",
  chunk_size = 1000L
)
```

Arguments

`client`	A `harbour_client`.
`table`	Table name.
`data`	A tibble or data frame whose columns match the table schema.
`...`	These dots are for future extensions and must be empty.
`row_id_col`	Name of the column in `data` that holds row IDs. Default `"_id"`.
`chunk_size`	Rows per request. SeaTable caps batch writes at 1000 and harbouR clamps it, warning if you asked for more.

Value

Invisibly, a one-row tibble with columns `table` (chr), `n_rows` (int) and `n_requests` (int).

See Also

Other rows: `hb_append_rows()`, `hb_delete_rows()`, `hb_get_row()`, `hb_lock_rows()`, `hb_query()`, `hb_read_table()`, `hb_unlock_rows()`

Examples

```
## Not run:
client <- hb_client()
hb_update_rows(
  client, "Samples",
  tibble::tibble(`_id` = "abc", Name = "renamed")
)

## End(Not run)
```

`hb_update_select_option` *Update a single-select option*

Description

Update a single-select option

Usage

```
hb_update_select_option(client, table, name, option, new_option, ...)
```

Arguments

`client`	A `harbour_client`.
`table`	Table name.
`name`	Column name.
`option`	Name of the option to rename.
`new_option`	New option name.
`...`	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other columns: `hb_add_column()`, `hb_add_columns()`, `hb_add_select_option()`, `hb_delete_column()`, `hb_delete_select_option()`, `hb_list_columns()`, `hb_update_column()`

Examples

```
## Not run:
client <- hb_client()
hb_update_select_option(client, "Samples", "Status", "Done", "Complete")

## End(Not run)
```

hb_update_view *Update a view*

Description

SeaTable's API addresses one view by its name in the URL, and its server reads a slash in that name as a path separator and a name of one or two dots as a folder. Such a view cannot be changed through the API, and harbouR says so before sending anything: rename it in SeaTable first.

Usage

```
hb_update_view(client, table, view, settings, ...)
```

Arguments

client	A harbour_client.
table	Table name.
view	Name of the view to update.
settings	Named list of the fields to change: name to rename the view, is_locked, filters, filter_conjunction, sorts, groupbys or hidden_columns. Fields you leave out are unchanged. hidden_columns takes column keys - the key column of hb_list_columns() - not names.
...	These dots are for future extensions and must be empty.

Value

Invisibly returns the client.

See Also

Other views: hb_create_view(), hb_delete_view(), hb_get_view(), hb_list_views()

Examples

```
## Not run:
client <- hb_client()
hb_update_view(client, "Samples", "Active", list(filter_conjunction = "And"))

## End(Not run)
```

`hb_upload_file` *Upload a file to SeaTable*

Description

Uploads `path` to the active base and returns the SeaTable file object (a named list with `name`, `size`, `type` and `url`) ready to be written into a `file`-typed cell via `hb_update_rows()` or `hb_attach_file()`.

Usage

```
hb_upload_file(client, path, ..., relative_path = "files")
```

Arguments

`client`	A `harbour_client`.
`path`	Local file path. Must exist.
`...`	These dots are for future extensions and must be empty.
`relative_path`	Folder under the base's asset root. The default, `"files"`, stands for the folder the upload link names: `files/<YYYY-MM>` for the current month, or `images/<YYYY-MM>` when `path` has an image extension (`.png`, `.jpg`, `.jpeg`, `.gif`, `.bmp`, `.webp` or `.svg`). Passing `"files"` yourself does the same. Any other value is used as given, but may not climb out of the asset root with `..`.

Details

SeaTable's file endpoints authenticate with the base's API token, so this needs a client created with `api_token`; a username/password client gets a `harbour_error_auth` condition.

A finite client timeout bounds the complete upload request. A timeout can occur after the server has accepted the file; it does not establish that the upload was rolled back. Check the server before retrying to avoid creating a duplicate attachment. Uploads are not retried automatically.

Value

A named list describing the uploaded asset.

See Also

Other files: `hb_attach_file()`, `hb_delete_asset()`, `hb_download_file()`

Examples

```
## Not run:
client <- hb_client()
hb_upload_file(client, "report.pdf")

## End(Not run)
```

`hb_validate_dtable` *Check a base for problems before writing it*

Description

Reports structural issues that would make SeaTable reject the file, or that indicate the base was built incorrectly in R: missing top-level keys; a table without a name or an `_id`; repeated table names, table ids, column keys, column names or row ids; a column without a key, a type or a name, or one named `_id`; rows without an `_id`; and empty table or view settings that would be written as `[]` where SeaTable expects `{}`.

Usage

```
hb_validate_dtable(x, ...)
```

Arguments

`x`	A `harbour_dtable`.
`...`	These dots are for future extensions and must be empty.

Value

A tibble with columns `location` (chr), `problem` (chr) and `severity` (chr, `"error"` or `"warning"`). Zero rows means the base is well-formed.

See Also

Other dtable: `as_tibble.harbour_dtable()`, `hb_asset_path()`, `hb_dtable()`, `hb_read_csv()`, `hb_read_dtable()`, `hb_read_xlsx()`, `hb_write_csv()`, `hb_write_dtable()`, `hb_write_xlsx()`, `is_harbour_dtable()`, `length.harbour_dtable()`, `names.harbour_dtable()`, `print.harbour_dtable()`, `summary.harbour_dtable()`

Examples

```
base <- hb_dtable(Samples = data.frame(x = 1))
hb_validate_dtable(base)
```

`hb_write_csv` *Export a base to CSV files*

Description

Writes one `.csv` per table into `dir`. Subject to the same losses as `hb_write_xlsx()`; see its documentation. Each file is named after its table, with the characters a file name cannot hold on some system - `/ \ : * ? " < > |` and control characters - replaced by `_`. Table names that produce the same filename (ignoring case) are rejected before any files are written.

Usage

```
hb_write_csv(x, dir, ..., tables = NULL)
```

Arguments

`x`	A `harbour_dtable`, or a `harbour_client` to read from first.
`dir`	Destination directory. Created if it does not exist.
`...`	These dots are for future extensions and must be empty.
`tables`	Table names to export. Default: all of them.

Value

A character vector of the files written, invisibly.

See Also

`hb_write_xlsx()`, `hb_write_dtable()`

Other dtable: `as_tibble.harbour_dtable()`, `hb_asset_path()`, `hb_dtable()`, `hb_read_csv()`, `hb_read_dtable()`, `hb_read_xlsx()`, `hb_validate_dtable()`, `hb_write_dtable()`, `hb_write_xlsx()`, `is_harbour_dtable()`, `length.harbour_dtable()`, `names.harbour_dtable()`, `print.harbour_dtable()`, `summary.harbour_dtable()`

Examples

```
base <- hb_read_dtable(
  system.file("extdata", "example.dtable", package = "harbouR")
)
dir <- tempfile()
hb_write_csv(base, dir)
list.files(dir)
```

hb_write_dtable *Write a* harbour_dtable *back to a* .dtable *file*

Description

Produces a ZIP archive SeaTable can import. Round-tripping is lossless: reading a file and writing it back yields a `content.json` that parses to an identical structure, verified against a real 750 KB export. The attachments go back in too when the file was read with `assets = "extract"`.

Usage

```
hb_write_dtable(x, path, ..., assets = TRUE, overwrite = TRUE)
```

Arguments

x	A harbour_dtable.
path	Destination path.
...	These dots are for future extensions and must be empty.
assets	Whether to repack the extracted asset tree, if there is one. Default TRUE.
overwrite	Whether to replace an existing file at path. Default TRUE; set FALSE to refuse.

Details

The archive is built in a temporary file beside path and moved into place only once it is complete, so a failed write cannot destroy the file it replaces - even when that is the file the base was read from. Replacing a file gives it a new identity on disk, so any other hard link to it keeps the old content. Where no new file can be made beside path, or there is nothing at path to lose - an empty file such as /dev/null, or a link to a file not made yet - the archive is written straight into it. A file that cannot be replaced, because another program holds it open or the folder lets files be written but not removed, is written into once the archive is complete; that needs room for a second copy, and should it fail, the complete archive is kept beside path and the error says where. A symbolic link in the extracted asset tree is left out, with a warning, rather than followed to wherever it points, including when asset/ itself is a link. Recorded attachments whose extracted files are missing are left out with a warning of class harbour_warning_missing_assets naming them.

Value

x, invisibly.

See Also

hb_read_dtable()

Other dtable: as_tibble.harbour_dtable(), hb_asset_path(), hb_dtable(), hb_read_csv(), hb_read_dtable(), hb_read_xlsx(), hb_validate_dtable(), hb_write_csv(), hb_write_xlsx(), is_harbour_dtable(), length.harbour_dtable(), names.harbour_dtable(), print.harbour_dtable(), summary.harbour_dtable()

Examples

```
path <- system.file("extdata", "example.dtable", package = "harbouR")
# extract the assets, so the attachments are written back too
base <- hb_read_dtable(path, assets = "extract")

out <- tempfile(fileext = ".dtable")
hb_write_dtable(base, out)

# content.json round-trips exactly
identical(hb_read_dtable(out)$content, base$content)
```

`hb_write_xlsx` *Export a base to an Excel workbook*

Description

Writes one worksheet per table. This is deliberately **one-way**: a spreadsheet cell holds a single value, so several SeaTable column types have no faithful representation.

Usage

```
hb_write_xlsx(x, path, ..., tables = NULL)
```

Arguments

`x`	A `harbour_dtable`, or a `harbour_client` to read from first.
`path`	Destination `.xlsx` path.
`...`	These dots are for future extensions and must be empty.
`tables`	Table names to export. Default: all of them.

Details

Each sheet is named after its table, made to fit Excel's rules for sheet names: cut to 31 characters, with each of `/`, `\`, `?`, `*`, `:` or a square bracket replaced by `_`. Excel compares sheet names ignoring case, so table names that would give sheet names differing only in the case of the letters A to Z, such as `Samples` and `samples`, are rejected before anything is written. So is text longer than the 32,767 characters a workbook cell holds.

Value

`path`, invisibly.

What is lost

- `multiple-select`, `collaborator` and `image` cells are joined with `", "`.
- `file` cells are reduced to their file names.
- `link` and `link-formula` cells reference rows in other tables and are exported as their display values.
- `formula` columns export their computed value, not the formula.
- `geolocation` cells are written as their values, in the order the source stores them, without the field names: `"11.5761, 48.1371"` does not say which number is the latitude.
- `digital-sign` cells are written as a placeholder, `"<object>"`, or as their bare values, depending on how the source stores them; the signature itself is not exported. `button` columns hold no value and are written as missing.
- A CSV stores numbers with 15 significant digits and a workbook with 16, so a double that needs more, such as `0.1 + 0.2`, reads back very slightly different. In a CSV, date-times are written by `as.character()`, whose layout depends on the version of R. Use `hb_write_dtable()` for full precision.
- Views, filters, colours, column widths and every other piece of base configuration have no cell to live in.

harbouR reports which columns were flattened. To keep everything, write a `.dtable` with `hb_write_dtable()` instead.

See Also

`hb_write_csv()`, `hb_write_dtable()`

Other dtable: `as_tibble.harbour_dtable()`, `hb_asset_path()`, `hb_dtable()`, `hb_read_csv()`, `hb_read_dtable()`, `hb_read_xlsx()`, `hb_validate_dtable()`, `hb_write_csv()`, `hb_write_dtable()`, `is_harbour_dtable()`, `length.harbour_dtable()`, `names.harbour_dtable()`, `print.harbour_dtable()`, `summary.harbour_dtable()`

Examples

```
base <- hb_read_dtable(
  system.file("extdata", "example.dtable", package = "harbouR")
)
out <- tempfile(fileext = ".xlsx")
hb_write_xlsx(base, out)
```

`is_harbour_client` *Test whether an object is a harbour client*

Description

Test whether an object is a harbour client

Usage

```
is_harbour_client(x)
```

Arguments

x	Object to test.

Value

A single TRUE or FALSE.

See Also

Other client: hb_check_credentials(), hb_client(), hb_ping(), hb_server_info()

Examples

```
# creating a client does not contact the server
client <- hb_client(
  server = "https://seatable.example.org",
  api_token = "your-api-token"
)
is_harbour_client(client)
is_harbour_client(list())
```

---

is_harbour_dtable *Test whether an object is a harbour dtable*

Description

Test whether an object is a harbour dtable

Usage

```
is_harbour_dtable(x)
```

Arguments

x	Object to test.

Value

A single TRUE or FALSE.

See Also

Other dtable: as_tibble.harbour_dtable(), hb_asset_path(), hb_dtable(), hb_read_csv(), hb_read_dtable(), hb_read_xlsx(), hb_validate_dtable(), hb_write_csv(), hb_write_dtable(), hb_write_xlsx(), length.harbour_dtable(), names.harbour_dtable(), print.harbour_dtable(), summary.harbour_dtable()

Examples

```
is_harbour_dtable(hb_dtable(Samples = data.frame(x = 1)))
```

---

is_harbour_metadata *Test whether an object is harbour metadata*

Description

Test whether an object is harbour metadata

Usage

```
is_harbour_metadata(x)
```

Arguments

x	Object to test.

Value

A single TRUE or FALSE.

See Also

Other metadata: as_tibble.harbour_metadata(), hb_column_types(), hb_list_collaborators(), hb_list_tables(), hb_metadata(), print.harbour_metadata(), summary.harbour_metadata()

Examples

```
is_harbour_metadata(hb_example_metadata())
is_harbour_metadata(list())
```

---

length.harbour_dtable *Number of tables in a dtable*

Description

Number of tables in a dtable

Usage

```
## S3 method for class 'harbour_dtable'
length(x)
```

Arguments

x	A harbour_dtable.

Value

A single integer.

See Also

Other dtable: as_tibble.harbour_dtable(), hb_asset_path(), hb_dtable(), hb_read_csv(), hb_read_dtable(), hb_read_xlsx(), hb_validate_dtable(), hb_write_csv(), hb_write_dtable(), hb_write_xlsx(), is_harbour_dtable(), names.harbour_dtable(), print.harbour_dtable(), summary.harbour_dtable()

Examples

```
length(hb_dtable(Samples = data.frame(x = 1)))
```

---

names.harbour_dtable *Names of the tables in a dtable*

Description

Names of the tables in a dtable

Usage

```
## S3 method for class 'harbour_dtable'
names(x)
```

Arguments

x	A harbour_dtable.

Value

A character vector of table names.

See Also

Other dtable: as_tibble.harbour_dtable(), hb_asset_path(), hb_dtable(), hb_read_csv(), hb_read_dtable(), hb_read_xlsx(), hb_validate_dtable(), hb_write_csv(), hb_write_dtable(), hb_write_xlsx(), is_harbour_dtable(), length.harbour_dtable(), print.harbour_dtable(), summary.harbour_dtable()

Examples

```
names(hb_dtable(Samples = data.frame(x = 1), Notes = data.frame(y = 2)))
```

print.harbour_dtable *Print a harbour dtable*

Description

Print a harbour dtable

Usage

```
## S3 method for class 'harbour_dtable'
print(x, ...)

## S3 method for class 'harbour_dtable'
format(x, ...)
```

Arguments

x	A harbour_dtable.
...	These dots are for future extensions and must be empty.

Value

print() returns x, invisibly.

format() returns a character vector, one element per table.

See Also

Other dtable: as_tibble.harbour_dtable(), hb_asset_path(), hb_dtable(), hb_read_csv(), hb_read_dtable(), hb_read_xlsx(), hb_validate_dtable(), hb_write_csv(), hb_write_dtable(), hb_write_xlsx(), is_harbour_dtable(), length.harbour_dtable(), names.harbour_dtable(), summary.harbour_dtable()

Examples

```
print(hb_dtable(Samples = data.frame(x = 1)))
```

print.harbour_metadata *Print method for harbour metadata*

Description

Print method for harbour metadata

Usage

```
## S3 method for class 'harbour_metadata'
print(x, ...)

## S3 method for class 'harbour_metadata'
format(x, ...)
```

Arguments

x	A harbour_metadata object.
...	These dots are for future extensions and must be empty.

Value

print() returns x, invisibly.

format() returns a character vector, one element per table.

See Also

Other metadata: as_tibble.harbour_metadata(), hb_column_types(), hb_list_collaborators(), hb_list_tables(), hb_metadata(), is_harbour_metadata(), summary.harbour_metadata()

Examples

```
meta <- hb_example_metadata()
print(meta)
format(meta)
```

---

summary.harbour_dtable

Summarise the schema of a dtable

Description

Summarise the schema of a dtable

Usage

```
## S3 method for class 'harbour_dtable'
summary(object, ...)
```

Arguments

object	A harbour_dtable.
...	These dots are for future extensions and must be empty.

Value

A tibble with one row per column across all tables: `table` (chr), `column` (chr), `type` (chr), `key` (chr).

See Also

Other dtable: `as_tibble.harbour_dtable()`, `hb_asset_path()`, `hb_dtable()`, `hb_read_csv()`, `hb_read_dtable()`, `hb_read_xlsx()`, `hb_validate_dtable()`, `hb_write_csv()`, `hb_write_dtable()`, `hb_write_xlsx()`, `is_harbour_dtable()`, `length.harbour_dtable()`, `names.harbour_dtable()`, `print.harbour_dtable()`

Examples

```
summary(hb_dtable(Samples = data.frame(x = 1, y = "a")))
```

---

summary.harbour_metadata

Summary of harbour metadata

Description

Summary of harbour metadata

Usage

```
## S3 method for class 'harbour_metadata'
summary(object, ...)
```

Arguments

`object`	A `harbour_metadata` object.
`...`	These dots are for future extensions and must be empty.

Value

A tibble with one row per column across all tables, suitable for inspecting the SeaTable schema. Columns: `table` (chr), `column` (chr), `type` (chr).

See Also

Other metadata: `as_tibble.harbour_metadata()`, `hb_column_types()`, `hb_list_collaborators()`, `hb_list_tables()`, `hb_metadata()`, `is_harbour_metadata()`, `print.harbour_metadata()`

Examples

```
summary(hb_example_metadata())
```

Index

∗ **client**
hb_check_credentials, 12
hb_client, 13
hb_ping, 35
hb_server_info, 44
is_harbour_client, 55
∗ **columns**
hb_add_column, 7
hb_add_columns, 8
hb_add_select_option, 9
hb_delete_column, 19
hb_delete_select_option, 20
hb_list_columns, 30
hb_update_column, 46
hb_update_select_option, 48
∗ **dtable**
as_tibble.harbour_dtable, 3
hb_asset_path, 11
hb_dtable, 24
hb_read_csv, 37
hb_read_dtable, 38
hb_read_xlsx, 41
hb_validate_dtable, 51
hb_write_csv, 52
hb_write_dtable, 53
hb_write_xlsx, 54
is_harbour_dtable, 56
length.harbour_dtable, 57
names.harbour_dtable, 58
print.harbour_dtable, 59
summary.harbour_dtable, 60
∗ **example data**
hb_example_metadata, 26
hb_example_rows, 27
∗ **files**
hb_attach_file, 11
hb_delete_asset, 18
hb_download_file, 23
hb_upload_file, 50
∗ **metadata**
as_tibble.harbour_metadata, 4
hb_column_types, 15
hb_list_collaborators, 30
hb_list_tables, 32
hb_metadata, 35
is_harbour_metadata, 57
print.harbour_metadata, 59
summary.harbour_metadata, 61
∗ **rows**
hb_append_rows, 10
hb_delete_rows, 19
hb_get_row, 28
hb_lock_rows, 34
hb_query, 36
hb_read_table, 40
hb_unlock_rows, 45
hb_update_rows, 47
∗ **shiny**
hb_run_explorer, 43
∗ **tables**
hb_create_table, 16
hb_delete_table, 21
hb_duplicate_table, 25
hb_rename_table, 42
∗ **views**
hb_create_view, 17
hb_delete_view, 22
hb_get_view, 29
hb_list_views, 33
hb_update_view, 49

as.Date(), *15*
as.logical(), *6*
as_tibble.harbour_dtable, 3
as_tibble.harbour_dtable(), *11*, *24*, *38*, *40*, *42*, *51–53*, *55*, *56*, *58*, *59*, *61*
as_tibble.harbour_metadata, 4
as_tibble.harbour_metadata(), *16*, *30*, *32*, *35*, *57*, *60*, *61*

format.harbour_client *(*hb_client*)*, 13
format.harbour_dtable
*(*print.harbour_dtable*)*, 59
format.harbour_metadata
*(*print.harbour_metadata*)*, 59

harbouR-conditions, 5
hb_add_column, 7
hb_add_column(), *8, 9*, *19*, *21*, *31*, *46*, *48*
hb_add_columns, 8
hb_add_columns(), *7*, *9*, *19*, *21*, *31*, *46*, *48*
hb_add_select_option, 9
hb_add_select_option(), *7, 8*, *19*, *21*, *31*, *46*, *48*
hb_append_rows, 10
hb_append_rows(), *6*, *20*, *28*, *34*, *37*, *41*, *45*, *47*
hb_asset_path, 11
hb_asset_path(), *4*, *24*, *38–40*, *42*, *51–53*, *55, 56*, *58, 59*, *61*
hb_attach_file, 11
hb_attach_file(), *18*, *23*, *50*
hb_check_credentials, 12
hb_check_credentials(), *14*, *35, 36*, *45*, *56*
hb_client, 13
hb_client(), *13*, *36*, *44, 45*, *56*
hb_column_types, 15
hb_column_types(), *4*, *30–32*, *35*, *57*, *60, 61*
hb_create_table, 16
hb_create_table(), *8*, *21*, *25*, *43*
hb_create_view, 17
hb_create_view(), *22*, *29*, *33*, *49*
hb_delete_asset, 18
hb_delete_asset(), *12*, *23*, *50*
hb_delete_column, 19
hb_delete_column(), *7–9*, *21*, *31*, *46*, *48*
hb_delete_rows, 19
hb_delete_rows(), *10*, *28*, *34*, *37*, *41*, *45*, *47*
hb_delete_select_option, 20
hb_delete_select_option(), *7–9*, *19*, *31*, *46*, *48*
hb_delete_table, 21
hb_delete_table(), *17*, *25*, *43*
hb_delete_view, 22
hb_delete_view(), *17*, *29*, *33*, *49*
hb_download_file, 23
hb_download_file(), *12*, *18*, *50*
hb_dtable, 24
hb_dtable(), *4*, *6*, *11*, *38*, *40*, *42*, *51–53*, *55, 56*, *58, 59*, *61*
hb_duplicate_table, 25
hb_duplicate_table(), *17*, *21*, *43*
hb_example_metadata, 26
hb_example_metadata(), *27*
hb_example_rows, 27
hb_example_rows(), *26*
hb_get_row, 28
hb_get_row(), *10*, *20*, *34*, *37*, *41*, *45*, *47*
hb_get_view, 29
hb_get_view(), *17*, *22*, *33*, *49*
hb_list_collaborators, 30
hb_list_collaborators(), *4*, *16*, *32*, *35*, *57*, *60, 61*
hb_list_columns, 30
hb_list_columns(), *7–9*, *19*, *21*, *38*, *46*, *48, 49*
hb_list_tables, 32
hb_list_tables(), *4*, *16*, *30*, *35*, *38*, *57*, *60, 61*
hb_list_views, 33
hb_list_views(), *17*, *22*, *29*, *49*
hb_lock_rows, 34
hb_lock_rows(), *10*, *20*, *28*, *37*, *41*, *45*, *47*
hb_metadata, 35
hb_metadata(), *4*, *16*, *30*, *32*, *57*, *60, 61*
hb_ping, 35
hb_ping(), *13, 14*, *45*, *56*
hb_query, 36
hb_query(), *6*, *10*, *20*, *28*, *34*, *41*, *45*, *47*
hb_read_csv, 37
hb_read_csv(), *4*, *6*, *11*, *24*, *40–42*, *51–53*, *55, 56*, *58, 59*, *61*
hb_read_dtable, 38
hb_read_dtable(), *4*, *6*, *11*, *24*, *27*, *38*, *42*, *44*, *51–53*, *55, 56*, *58, 59*, *61*
hb_read_table, 40
hb_read_table(), *6*, *10*, *20*, *24*, *28*, *32*, *34*, *37, 38*, *45*, *47*
hb_read_xlsx, 41
hb_read_xlsx(), *4*, *6*, *11*, *24*, *38*, *40*, *51–53*, *55, 56*, *58, 59*, *61*
hb_rename_table, 42
hb_rename_table(), *17*, *21*, *25*
hb_run_explorer, 43
hb_server_info, 44
hb_server_info(), *13, 14*, *36*, *56*

hb_unlock_rows, 45
hb_unlock_rows(), *10*, *20*, *28*, *34*, *37*, *41*, *47*
hb_update_column, 46
hb_update_column(), *7–9*, *19*, *21*, *31*, *48*
hb_update_rows, 47
hb_update_rows(), *6*, *10*, *12*, *20*, *28*, *34*, *37*, *41*, *45*, *50*
hb_update_select_option, 48
hb_update_select_option(), *7–9*, *19*, *21*, *31*, *46*
hb_update_view, 49
hb_update_view(), *17*, *22*, *29*, *33*
hb_upload_file, 50
hb_upload_file(), *12*, *18*, *23*
hb_validate_dtable, 51
hb_validate_dtable(), *4*, *11*, *24*, *38*, *40*, *42*, *52, 53*, *55, 56*, *58, 59*, *61*
hb_write_csv, 52
hb_write_csv(), *4*, *11*, *24*, *37, 38*, *40*, *42*, *51*, *53*, *55, 56*, *58, 59*, *61*
hb_write_dtable, 53
hb_write_dtable(), *4*, *6*, *11*, *24*, *38–40*, *42*, *51, 52*, *55, 56*, *58, 59*, *61*
hb_write_xlsx, 54
hb_write_xlsx(), *4*, *11*, *24*, *38*, *40*, *42*, *51–53*, *56*, *58, 59*, *61*

is_harbour_client, 55
is_harbour_client(), *13, 14*, *36*, *45*
is_harbour_dtable, 56
is_harbour_dtable(), *4*, *11*, *24*, *38*, *40*, *42*, *51–53*, *55*, *58, 59*, *61*
is_harbour_metadata, 57
is_harbour_metadata(), *4*, *16*, *30*, *32*, *35*, *60, 61*

length.harbour_dtable, 57
length.harbour_dtable(), *4*, *11*, *24*, *38*, *40*, *42*, *51–53*, *55, 56*, *58, 59*, *61*

names.harbour_dtable, 58
names.harbour_dtable(), *4*, *11*, *24*, *38*, *40*, *42*, *51–53*, *55, 56*, *58, 59*, *61*

print.harbour_client *(*hb_client*)*, 13
print.harbour_dtable, 59
print.harbour_dtable(), *4*, *11*, *24*, *38*, *40*, *42*, *51–53*, *55, 56*, *58*, *61*
print.harbour_metadata, 59
print.harbour_metadata(), *4*, *16*, *30*, *32*, *35*, *57*, *61*

shiny::runApp(), *43*
summary.harbour_dtable, 60
summary.harbour_dtable(), *4*, *11*, *24*, *38*, *40*, *42*, *51–53*, *55, 56*, *58, 59*
summary.harbour_metadata, 61
summary.harbour_metadata(), *4*, *16*, *30*, *32*, *35*, *57*, *60*

utils::read.csv(), *38*

zip::zip_list(), *39*