Package 'adbcdrivermanager'

March 9, 2026

Title 'Arrow' Database Connectivity ('ADBC') Driver Manager

Version 0.22.0-1

Description Provides a developer-facing interface to 'Arrow' Database Connectivity ('ADBC') for the purposes of driver development, driver testing, and building high-level database interfaces for users. 'ADBC' <https://arrow.apache.org/adbc/> is an API standard for database access libraries that uses 'Arrow' for result sets and query parameters.

License Apache License (>= 2)

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/build/bootstrap TRUE

URL https://arrow.apache.org/adbc/current/r/adbcdrivermanager/, https://github.com/apache/arrow-adbc

BugReports https://github.com/apache/arrow-adbc/issues

Imports nanoarrow (>= 0.3.0)

NeedsCompilation yes

Author Dewey Dunnington [aut, cre] (ORCID: <https://orcid.org/0000-0002-9415-4582>), Apache Arrow [aut, cph], Apache Software Foundation [cph]

Maintainer Dewey Dunnington <dewey@dunnington.ca>

Repository CRAN

Date/Publication 2026-03-09 10:10:02 UTC

Contents

`adbc_connection_get_info` *Connection methods*

Description

Connection methods

Usage

```
adbc_connection_get_info(connection, info_codes = NULL)

adbc_connection_get_objects(
  connection,
  depth = 0L,
  catalog = NULL,
  db_schema = NULL,
  table_name = NULL,
  table_type = NULL,
  column_name = NULL
)

adbc_connection_get_table_schema(connection, catalog, db_schema, table_name)

adbc_connection_get_table_types(connection)

adbc_connection_read_partition(connection, serialized_partition)
```

```
adbc_connection_commit(connection)

adbc_connection_rollback(connection)

adbc_connection_cancel(connection)

adbc_connection_get_statistic_names(connection)

adbc_connection_get_statistics(
  connection,
  catalog,
  db_schema,
  table_name,
  approximate = FALSE
)

adbc_connection_quote_identifier(connection, value, ...)

adbc_connection_quote_string(connection, value, ...)
```

Arguments

connection	An adbc_connection
info_codes	A list of metadata codes to fetch, or NULL to fetch all. Valid values are documented in the adbc.h header.
depth	The level of nesting to display. If 0, display all levels. If 1, display only catalogs (i.e., catalog_schemas will be null). If 2, display only catalogs and schemas (i.e., db_schema_tables will be null). If 3, display only catalogs, schemas, and tables.
catalog	Only show tables in the given catalog. If NULL, do not filter by catalog. If an empty string, only show tables without a catalog. May be a search pattern.
db_schema	Only show tables in the given database schema. If NULL, do not filter by database schema. If an empty string, only show tables without a database schema. May be a search pattern.
table_name	Constrain an object or statistics query for a specific table. If NULL, do not filter by name. May be a search pattern.
table_type	Only show tables matching one of the given table types. If NULL, show tables of any type. Valid table types can be fetched from GetTableTypes. Terminate the list with a NULL entry.
column_name	Only show columns with the given name. If NULL, do not filter by name. May be a search pattern.
serialized_partition	The partition descriptor.
approximate	If FALSE, request exact values of statistics, else allow for best-effort, approximate, or cached values. The database may return approximate values regardless, as indicated in the result. Requesting exact values may be expensive or unsupported.

value	A string or identifier.
...	Driver-specific options. For the default method, these are named values that are converted to strings.

Value

- adbc_connection_get_info(), adbc_connection_get_objects(), adbc_connection_get_table_types(), and adbc_connection_read_partition() return a nanoarrow_array_stream.
- adbc_connection_get_table_schema() returns a nanoarrow_schena
- adbc_connection_commit() and adbc_connection_rollback() return connection, invisibly.

Examples

```
db <- adbc_database_init(adbc_driver_void())
con <- adbc_connection_init(db)
# (not implemented by the void driver)
try(adbc_connection_get_info(con, 0))
```

---

adbc_connection_init *Connections*

Description

Connections

Usage

```
adbc_connection_init(database, ...)

adbc_connection_init_default(database, options = NULL, subclass = character())

adbc_connection_release(connection)

adbc_connection_set_options(connection, options)

adbc_connection_get_option(connection, option)

adbc_connection_get_option_bytes(connection, option)

adbc_connection_get_option_int(connection, option)

adbc_connection_get_option_double(connection, option)
```

Arguments

database	An adbc_database.
...	Driver-specific options. For the default method, these are named values that are converted to strings.
options	A named character() or list() whose values are converted to strings.
subclass	An extended class for an object so that drivers can specify finer-grained control over behaviour at the R level.
connection	An adbc_connection
option	A specific option name

Value

An object of class 'adbc_connection'

Examples

```
db <- adbc_database_init(adbc_driver_void())
adbc_connection_init(db)
```

---

adbc_connection_join *Join the lifecycle of a unique parent to its child*

Description

It is occasionally useful to return a connection, statement, or stream from a function that was created from a unique parent. These helpers tie the lifecycle of a unique parent object to its child such that the parent object is released predictably and immediately after the child. These functions will invalidate all references to the previous R object.

Usage

```
adbc_connection_join(connection, database)

adbc_statement_join(statement, connection)
```

Arguments

connection	A connection created with adbc_connection_init()
database	A database created with adbc_database_init()
statement	A statement created with adbc_statement_init()

Value

The input, invisibly.

Examples

```
# Use local_adbc to ensure prompt cleanup on error;
# use join functions to return a single object that manages
# the lifecycle of all three.
stmt <- local({
  db <- local_adbc(adbc_database_init(adbc_driver_log()))

  con <- local_adbc(adbc_connection_init(db))
  adbc_connection_join(con, db)

  stmt <- local_adbc(adbc_statement_init(con))
  adbc_statement_join(stmt, con)

  adbc_xptr_move(stmt)
})

# Everything is released immediately when the last object is released
adbc_statement_release(stmt)
```

---

adbc_database_init *Databases*

Description

Databases

Usage

```
adbc_database_init(driver, ...)

adbc_database_init_default(driver, options = NULL, subclass = character())

adbc_database_release(database)

adbc_database_set_options(database, options)

adbc_database_get_option(database, option)

adbc_database_get_option_bytes(database, option)

adbc_database_get_option_int(database, option)

adbc_database_get_option_double(database, option)
```

Arguments

driver	An adbc_driver().
...	Driver-specific options. For the default method, these are named values that are converted to strings.
options	A named character() or list() whose values are converted to strings.
subclass	An extended class for an object so that drivers can specify finer-grained control over behaviour at the R level.
database	An adbc_database.
option	A specific option name

Value

An object of class adbc_database

Examples

```
adbc_database_init(adbc_driver_void())
```

---

adbc_driver_load *Low-level driver loader*

Description

Most users should use adbc_driver(); however, this function may be used to allow other libraries (e.g., GDAL) to access the driver loader.

Usage

```
adbc_driver_load(
  x,
  entrypoint,
  version,
  driver,
  error,
  load_flags = adbc_load_flags(),
  additional_search_path_list = NULL
)
```

Arguments

x, entrypoint	An ADBC driver may be defined either as an init function or as an identifier with an entrypoint name. A driver init func must be an external pointer to a DL_FUNC with the type AdbcDriverInitFunc specified in the adbc.h header.
version	The version number corresponding to the driver supplied

`driver`	An external pointer to an `AdbcDriver`
`error`	An external pointer to an `AdbcError` or NULL
`load_flags`	Integer flags generated by `adbc_load_flags()`
`additional_search_path_list`	A path list of additional locations to search for driver manifests

Value

An integer ADBC status code

`adbc_driver_log` *Log calls to another driver*

Description

Useful for debugging or ensuring that certain calls occur during initialization and/or cleanup. The current logging output should not be considered stable and may change in future releases.

Usage

```
adbc_driver_log()
```

Value

An object of class 'adbc_driver_log'

Examples

```
drv <- adbc_driver_log()
db <- adbc_database_init(drv, key = "value")
con <- adbc_connection_init(db, key = "value")
stmt <- adbc_statement_init(con, key = "value")
try(adbc_statement_execute_query(stmt))
adbc_statement_release(stmt)
adbc_connection_release(con)
adbc_database_release(db)
```

adbc_driver_monkey *Monkey see, monkey do!*

Description

A driver whose query results are set in advance.

Usage

```
adbc_driver_monkey()
```

Value

An object of class 'adbc_driver_monkey'

Examples

```
db <- adbc_database_init(adbc_driver_monkey())
con <- adbc_connection_init(db)
stmt <- adbc_statement_init(con, mtcars)
stream <- nanoarrow::nanoarrow_allocate_array_stream()
adbc_statement_execute_query(stmt, stream)
as.data.frame(stream$get_next())
```

adbc_driver_void *Create ADBC drivers*

Description

Creates the R object representation of an ADBC driver, which consists of a name and an initializer function with an optional subclass to control finer-grained behaviour at the R level.

Usage

```
adbc_driver_void()

adbc_driver(
  x,
  entrypoint = NULL,
  ...,
  load_flags = adbc_load_flags(),
  subclass = character()
)
```

Arguments

x, entrypoint	An ADBC driver may be defined either as an init function or as an identifier with an entrypoint name. A driver init func must be an external pointer to a DL_FUNC with the type AdbcDriverInitFunc specified in the adbc.h header.
...	Further key/value parameters to store with the (R-level) driver object.
load_flags	Integer flags generated by adbc_load_flags()
subclass	An optional subclass for finer-grained control of behaviour at the R level.

Value

An object of class 'adbc_driver'

Examples

```
adbc_driver_void()
```

---

adbc_error_from_array_stream

Get extended error information from an array stream

Description

Get extended error information from an array stream

Usage

```
adbc_error_from_array_stream(stream)
```

Arguments

stream	A nanoarrow_array_stream

Value

NULL if stream was not created by a driver that supports extended error information or a list whose first element is the status code and second element is the adbc_error object. The acbc_error must not be accessed if stream is explicitly released.

Examples

```
db <- adbc_database_init(adbc_driver_monkey())
con <- adbc_connection_init(db)
stmt <- adbc_statement_init(con, mtcars)
stream <- nanoarrow::nanoarrow_allocate_array_stream()
adbc_statement_execute_query(stmt, stream)
adbc_error_from_array_stream(stream)
```

adbc_load_flags *Driver search/load options*

Description

Options that indicate where to look for driver manifests. Manifests (.toml files) can be installed at the system level, the user level, in location(s) specified by the ADBC_DRIVER_PATH environment variable, and/or in a conda environment. See the ADBC documentation for details regarding the locations of the user and system paths on various platforms.

Usage

```
adbc_load_flags(
  search_env = TRUE,
  search_user = TRUE,
  search_system = TRUE,
  allow_relative_paths = TRUE
)
```

Arguments

search_env	Search for manifest files in the directories specified in the ADBC_DRIVER_PATH environment variable and (when installed with conda) in the conda environment.
search_user	Search for manifest files in the designated directory for user ADBC driver installs.
search_system	Search for manifest files in the designtaed directory for system ADBC driver installs.
allow_relative_paths	Allow shared objects to be specified relative to the current working directory.

Value

An integer flag value for use in `adbc_driver()`

adbc_statement_init *Statements*

Description

Statements

Usage

```
adbc_statement_init(connection, ...)

adbc_statement_init_default(connection, options = NULL, subclass = character())

adbc_statement_release(statement)

adbc_statement_set_options(statement, options)

adbc_statement_get_option(statement, option)

adbc_statement_get_option_bytes(statement, option)

adbc_statement_get_option_int(statement, option)

adbc_statement_get_option_double(statement, option)
```

Arguments

`connection`	An adbc_connection
`...`	Driver-specific options. For the default method, these are named values that are converted to strings.
`options`	A named `character()` or `list()` whose values are converted to strings.
`subclass`	An extended class for an object so that drivers can specify finer-grained control over behaviour at the R level.
`statement`	An adbc_statement
`option`	A specific option name

Value

An object of class 'adbc_statement'

Examples

```
db <- adbc_database_init(adbc_driver_void())
con <- adbc_connection_init(db)
adbc_statement_init(con)
```

---

adbc_statement_set_sql_query

Statement methods

Description

Statement methods

Usage

```
adbc_statement_set_sql_query(statement, query)

adbc_statement_set_substrait_plan(statement, plan)

adbc_statement_prepare(statement)

adbc_statement_get_parameter_schema(statement)

adbc_statement_bind(statement, values, schema = NULL)

adbc_statement_bind_stream(statement, stream, schema = NULL)

adbc_statement_execute_query(
  statement,
  stream = NULL,
  stream_join_parent = FALSE
)

adbc_statement_execute_schema(statement)

adbc_statement_cancel(statement)
```

Arguments

`statement`	An adbc_statement
`query`	An SQL query as a string
`plan`	A raw vector representation of a serialized Substrait plan.
`values`	A nanoarrow_array or object that can be coerced to one.
`schema`	A nanoarrow_schema or object that can be coerced to one.
`stream`	A nanoarrow_array_stream or object that can be coerced to one.
`stream_join_parent`	Use `TRUE` to invalidate `statement` and tie its lifecycle to `stream`.

Value

- `adbc_statement_set_sql_query()`, `adbc_statement_set_substrait_plan()`, `adbc_statement_prepare()`, `adbc_statement_bind()`, `adbc_statement_bind_stream()`, and `adbc_statement_execute_query()` return `statement`, invisibly.
- `adbc_statement_get_parameter_schema()` returns a nanoarrow_schema.

Examples

```
db <- adbc_database_init(adbc_driver_void())
con <- adbc_connection_init(db)
stmt <- adbc_statement_init(con)
# (not implemented by the void driver)
```

```
try(adbc_statement_set_sql_query(stmt, "some query"))
```

`adbc_xptr_move` *Low-level pointer details*

Description

- `adbc_xptr_move()` allocates a fresh R object and moves all values pointed to by `x` into it. The original R object is invalidated by zeroing its content. This is useful when returning from a function where lifecycle helpers were used to manage the original object.
- `adbc_xptr_is_valid()` provides a means by which to test for an invalidated pointer.

Usage

```
adbc_xptr_move(x, check_child_count = TRUE)

adbc_xptr_is_valid(x)
```

Arguments

`x`	An 'adbc_database', 'adbc_connection', 'adbc_statement', or 'nanoarrow_array_stream'
`check_child_count`	Ensures that `x` has a zero child count before performing the move. This should almost always be `TRUE`.

Value

- `adbc_xptr_move()`: A freshly-allocated R object identical to `x`
- `adbc_xptr_is_valid()`: Returns FALSE if the ADBC object pointed to by `x` has been invalidated.

Examples

```
db <- adbc_database_init(adbc_driver_void())
adbc_xptr_is_valid(db)
db_new <- adbc_xptr_move(db)
adbc_xptr_is_valid(db)
adbc_xptr_is_valid(db_new)
```

read_adbc *Read, write, and execute on ADBC connections*

Description

These are convenience methods useful for testing connections. Note that S3 dispatch is always on db_or_con (i.e., drivers may provide their own implementations).

Usage

```
read_adbc(db_or_con, query, ..., bind = NULL)

execute_adbc(db_or_con, query, ..., bind = NULL)

write_adbc(
  tbl,
  db_or_con,
  target_table,
  ...,
  mode = c("default", "create", "append", "replace", "create_append"),
  temporary = FALSE,
  catalog_name = NULL,
  db_schema_name = NULL
)
```

Arguments

db_or_con	An adbc_database or adbc_connection. If a database, a connection will be opened. For read_adbc(), this connection will be closed when the resulting stream has been released.
query	An SQL query
...	Passed to S3 methods.
bind	A data.frame, nanoarrow_array, or nanoarrow_array_stream of bind parameters or NULL to skip the bind/prepare step.
tbl	A data.frame, nanoarrow_array, or nanoarrow_array_stream.
target_table	A target table name to which tbl should be written.
mode	One of "create", "append", "replace", "create_append" (error if the schema is not compatible or append otherwise), or "default" (use the adbc.ingest.mode argument of adbc_statement_init()). The default is "default".
temporary	Use TRUE to create a table as a temporary table.
catalog_name	If not NULL, the catalog to create/locate the table in.
db_schema_name	If not NULL, the schema to create/locate the table in.

Value

- `read_adbc()`: A nanoarrow_array_stream
- `execute_adbc()`: `db_or_con`, invisibly.
- `write_adbc()`: `tbl`, invisibly.

Examples

```
# On a database, connections are opened and closed
db <- adbc_database_init(adbc_driver_log())
try(read_adbc(db, "some sql"))
try(execute_adbc(db, "some sql"))
try(write_adbc(mtcars, db, "some_table"))

# Also works on a connection
con <- adbc_connection_init(db)
try(read_adbc(con, "some sql"))
try(execute_adbc(con, "some sql"))
try(write_adbc(mtcars, con, "some_table"))
```

---

with_adbc *Cleanup helpers*

Description

Managing the lifecycle of databases, connections, and statements can be complex and error-prone. The R objects that wrap the underlying ADBC pointers will perform cleanup in the correct order if you rely on garbage collection (i.e., do nothing and let the objects go out of scope); however it is good practice to explicitly clean up these objects. These helpers are designed to make explicit and predictable cleanup easy to accomplish.

Usage

```
with_adbc(x, code)

local_adbc(x, .local_envir = parent.frame())
```

Arguments

`x`	An ADBC database, ADBC connection, ADBC statement, or nanoarrow_array_stream returned from calls to an ADBC function.
`code`	Code to execute before cleaning up the input.
`.local_envir`	The execution environment whose scope should be tied to the input.

Details

Note that you can use adbc_connection_join() and adbc_statement_join() to tie the lifecycle of the parent object to that of the child object. These functions mark any previous references to the parent object as released so you can still use local and with helpers to manage the parent object before it is joined. Use stream_join_parent = TRUE in adbc_statement_execute_query() to tie the lifecycle of a statement to the output stream.

Value

- with_adbc() returns the result of code
- local_adbc() returns the input, invisibly.

Examples

```
# Using with_adbc():
with_adbc(db <- adbc_database_init(adbc_driver_void()), {
  with_adbc(con <- adbc_connection_init(db), {
    with_adbc(stmt <- adbc_statement_init(con), {
      # adbc_statement_set_sql_query(stmt, "SELECT * FROM foofy")
      # adbc_statement_execute_query(stmt)
      "some result"
    })
  })
})

# Using local_adbc_*() (works best within a function, test, or local())
local({
  db <- local_adbc(adbc_database_init(adbc_driver_void()))
  con <- local_adbc(adbc_connection_init(db))
  stmt <- local_adbc(adbc_statement_init(con))
  # adbc_statement_set_sql_query(stmt, "SELECT * FROM foofy")
  # adbc_statement_execute_query(stmt)
  "some result"
})
```

Index