Package 'Rduckhts'

February 23, 2026

Title 'DuckDB' High Throughput Sequencing File Formats Reader Extension

Version 0.1.2-0.1.4

Description Bundles the 'duckhts' 'DuckDB' extension for reading High Throughput Sequencing file formats with 'DuckDB'. The 'DuckDB' C extension API <https://duckdb.org/docs/stable/clients/c/api> and its 'htslib' dependency are compiled from vendored sources during package installation. James K Bonfield and co-authors (2021) <doi:10.1093/gigascience/giab007>.

License GPL-3

Copyright See inst/COPYRIGHT

Encoding UTF-8

SystemRequirements GNU make, cmake, zlib, libbz2, liblzma, libcurl, openssl (development headers)

Depends R (>= 4.4.0)

Imports DBI, duckdb, utils

Suggests tinytest

RoxygenNote 7.3.3

URL https://github.com/RGenomicsETL/duckhts

BugReports https://github.com/RGenomicsETL/duckhts/issues

NeedsCompilation no

Author Sounkou Mahamane Toure [aut, cre], James K Bonfield, John Marshall,Petr Danecek ,Heng Li , Valeriu Ohan, Andrew Whitwham,Thomas Keane , Robert M Davies [ctb] (Htslib Authors), DuckDB C Extension API Authors [ctb]

Maintainer Sounkou Mahamane Toure <sounkoutoure@gmail.com>

Repository CRAN

Date/Publication 2026-02-23 09:00:07 UTC

Contents

`detect_complex_types` *Detect Complex Types in DuckDB Table*

Description

Identifies columns in a DuckDB table that contain complex types (ARRAY or MAP) that will be returned as R lists.

Usage

```
detect_complex_types(con, table_name)
```

Arguments

`con`	A DuckDB connection
`table_name`	Name of the table to analyze

Value

A data frame with columns that have complex types, showing column_name, column_type, and a description of R type.

Examples

```
library(DBI)
library(duckdb)

con <- dbConnect(duckdb::duckdb(config = list(allow_unsigned_extensions = "true")))
rduckhts_load(con)
bcf_path <- system.file("extdata", "vcf_file.bcf", package = "Rduckhts")
rduckhts_bcf(con, "variants", bcf_path, overwrite = TRUE)
complex_cols <- detect_complex_types(con, "variants")
print(complex_cols)
dbDisconnect(con, shutdown = TRUE)
```

---

duckdb_type_mappings *DuckDB to R Type Mappings*

Description

The mapping covers the most common data types used in HTS file processing:

- BIGINT <-> double (not integer due to 64-bit overflow protection)
- DOUBLE <-> numeric/double
- VARCHAR <-> character/string
- BOOLEAN <-> logical
- ARRAY types (e.g., VARCHAR[], BIGINT[]) <-> list
- MAP types (e.g., MAP(VARCHAR, VARCHAR)) <-> data.frame

Important notes:

- 64-bit integers (BIGINT, UBIGINT) become double to prevent overflow
- DATE/TIME values return as Unix epoch numbers (double)
- MAP types become data frames with 'key' and 'value' columns
- ARRAY types become vectors (which are lists in R terminology)

Usage

```
duckdb_type_mappings()
```

Details

Returns a named list mapping between DuckDB and R data types. This is useful for understanding type conversions when reading HTS files or when specifying column types in tabix functions.

Value

A named list with two elements:

duckdb_to_r Named character vector mapping DuckDB types to R types

r_to_duckdb Named character vector mapping R types to DuckDB types

Examples

```
mappings <- duckdb_type_mappings()
mappings$duckdb_to_r["BIGINT"]
mappings$r_to_duckdb["integer"]
```

---

duckhts_bootstrap *Bootstrap the duckhts extension sources into the R package*

Description

Copies extension source files from the parent duckhts repository into `inst/duckhts_extension/` so the R package becomes self-contained. Run this before `R CMD build` to prepare the source tarball.

Usage

```
duckhts_bootstrap(repo_root = NULL)
```

Arguments

`repo_root`	Path to the duckhts repository root. Required.

Value

Invisibly returns the destination directory.

---

duckhts_build *Build the duckhts DuckDB extension*

Description

Compiles htslib and the duckhts extension from the sources bundled in the installed R package. The built `.duckdb_extension` file is placed in the extension directory.

Usage

```
duckhts_build(build_dir = NULL, make = NULL, force = FALSE, verbose = TRUE)
```

Arguments

build_dir	Where to build. Required. Use a writable location such as tempdir() when the installed package directory is read-only.
make	Optional GNU make command to use (e.g., "gmake" or "make"). When NULL, auto-detects gmake or make. If a non-GNU make is used, htslib's configure step will fail.
force	Rebuild even if the extension file already exists.
verbose	Print build output.

Value

Path to the built duckhts.duckdb_extension file.

---

duckhts_load *Load the duckhts extension into a DuckDB connection*

Description

Load the duckhts extension into a DuckDB connection

Usage

```
duckhts_load(con = NULL, extension_path = NULL)
```

Arguments

con	An existing DuckDB connection, or NULL to create one.
extension_path	Explicit path to the .duckdb_extension file. If NULL, uses the default location in the installed package.

Value

The DuckDB connection (invisibly).

extract_array_element *Extract Array Elements Safely*

Description

Helper function to safely extract elements from DuckDB arrays (returned as R lists) with proper error handling.

Usage

```
extract_array_element(array_col, index = NULL, default = NA)
```

Arguments

array_col	A list column from DuckDB array data
index	Numeric index (1-based). If NULL, returns full list
default	Default value if index is out of bounds

Value

The array element at the specified index, or full array if index is NULL

Examples

```
library(DBI)
library(duckdb)

con <- dbConnect(duckdb::duckdb(config = list(allow_unsigned_extensions = "true")))
rduckhts_load(con)
bcf_path <- system.file("extdata", "vcf_file.bcf", package = "Rduckhts")
rduckhts_bcf(con, "variants", bcf_path, overwrite = TRUE)
data <- dbGetQuery(con, "SELECT ALT FROM variants LIMIT 5")
first_alt <- extract_array_element(data$ALT, 1)
all_alts <- extract_array_element(data$ALT)
dbDisconnect(con, shutdown = TRUE)
```

extract_map_data *Extract MAP Keys and Values*

Description

Helper function to work with DuckDB MAP data (returned as data frames). Can extract keys, values, or search for specific key-value pairs.

Usage

```
extract_map_data(map_col, operation = "keys", default = NA)
```

Arguments

map_col	A data frame column from DuckDB MAP data
operation	What to extract: "keys", "values", or a specific key name
default	Default value if key is not found (only used when operation is a key name)

Value

Extracted data based on the operation

Examples

```
library(DBI)
library(duckdb)

con <- dbConnect(duckdb::duckdb(config = list(allow_unsigned_extensions = "true")))
rduckhts_load(con)
gff_path <- system.file("extdata", "gff_file.gff.gz", package = "Rduckhts")
rduckhts_gff(con, "annotations", gff_path, attributes_map = TRUE, overwrite = TRUE)
data <- dbGetQuery(con, "SELECT attributes FROM annotations LIMIT 5")
keys <- extract_map_data(data$attributes, "keys")
name_values <- extract_map_data(data$attributes, "Name")
dbDisconnect(con, shutdown = TRUE)
```

---

normalize_tabix_types *Normalize R Data Types to DuckDB Types for Tabix*

Description

Normalizes R data type names to their corresponding DuckDB types for use with tabix readers. This function handles common R type name variations and maps them to appropriate DuckDB column types.

Usage

```
normalize_tabix_types(types)
```

Arguments

types	A character vector of R data type names to be normalized.

Details

The function performs the following normalizations:

- Integer types (integer, int, int32, int64) -> BIGINT
- Numeric types (numeric, double, float) -> DOUBLE
- Character types (character, string, chr) -> VARCHAR
- Logical types (logical, bool, boolean) -> BOOLEAN
- Other types -> Converted to uppercase as-is

If an empty vector is provided, it returns the empty vector unchanged.

Value

A character vector of normalized DuckDB type names suitable for tabix columns.

See Also

`rduckhts_tabix` for using normalized types with tabix readers, `duckdb_type_mappings` for the complete type mapping table.

Examples

```
normalize_tabix_types(c("integer", "character", "numeric"))
normalize_tabix_types(c("int", "string", "float"))
```

---

rduckhts_bam *Create SAM/BAM/CRAM Table*

Description

Creates a DuckDB table from SAM, BAM, or CRAM files using the DuckHTS extension.

Usage

```
rduckhts_bam(
  con,
  table_name,
  path,
  region = NULL,
  reference = NULL,
  standard_tags = NULL,
  auxiliary_tags = NULL,
  overwrite = FALSE
)
```

Arguments

con	A DuckDB connection with DuckHTS loaded
table_name	Name for the created table
path	Path to the SAM/BAM/CRAM file
region	Optional genomic region (e.g., "chr1:1000-2000")
reference	Optional reference file path for CRAM files
standard_tags	Logical. If TRUE, include typed standard SAMtags columns
auxiliary_tags	Logical. If TRUE, include AUXILIARY_TAGS map of non-standard tags
overwrite	Logical. If TRUE, overwrites existing table

Value

Invisible TRUE on success

Examples

```
library(DBI)
library(duckdb)

con <- dbConnect(duckdb::duckdb(config = list(allow_unsigned_extensions = "true")))
rduckhts_load(con)
bam_path <- system.file("extdata", "range.bam", package = "Rduckhts")
rduckhts_bam(con, "reads", bam_path, overwrite = TRUE)
dbGetQuery(con, "SELECT COUNT(*) FROM reads WHERE FLAG & 4 = 0")
dbDisconnect(con, shutdown = TRUE)
```

---

rduckhts_bcf *Create VCF/BCF Table*

Description

Creates a DuckDB table from a VCF or BCF file using the DuckHTS extension. This follows the RBCFTools pattern of creating a table that can be queried.

Usage

```
rduckhts_bcf(
  con,
  table_name,
  path,
  region = NULL,
  tidy_format = FALSE,
  overwrite = FALSE
)
```

Arguments

con	A DuckDB connection with DuckHTS loaded
table_name	Name for the created table
path	Path to the VCF/BCF file
region	Optional genomic region (e.g., "chr1:1000-2000")
tidy_format	Logical. If TRUE, FORMAT columns are returned in tidy format
overwrite	Logical. If TRUE, overwrites existing table

Value

Invisible TRUE on success

Examples

```
library(DBI)
library(duckdb)

con <- dbConnect(duckdb::duckdb(config = list(allow_unsigned_extensions = "true")))
rduckhts_load(con)
bcf_path <- system.file("extdata", "vcf_file.bcf", package = "Rduckhts")
rduckhts_bcf(con, "variants", bcf_path, overwrite = TRUE)
dbGetQuery(con, "SELECT * FROM variants LIMIT 2")
dbDisconnect(con, shutdown = TRUE)
```

---

rduckhts_fasta *Create FASTA Table*

Description

Creates a DuckDB table from FASTA files using the DuckHTS extension.

Usage

```
rduckhts_fasta(con, table_name, path, overwrite = FALSE)
```

Arguments

con	A DuckDB connection with DuckHTS loaded
table_name	Name for the created table
path	Path to the FASTA file
overwrite	Logical. If TRUE, overwrites existing table

Value

Invisible TRUE on success

rduckhts_fastq *Create FASTQ Table*

Description

Creates a DuckDB table from FASTQ files using the DuckHTS extension.

Usage

```
rduckhts_fastq(
  con,
  table_name,
  path,
  mate_path = NULL,
  interleaved = FALSE,
  overwrite = FALSE
)
```

Arguments

con	A DuckDB connection with DuckHTS loaded
table_name	Name for the created table
path	Path to the FASTQ file
mate_path	Optional path to mate file for paired reads
interleaved	Logical indicating if file is interleaved paired reads
overwrite	Logical. If TRUE, overwrites existing table

Value

Invisible TRUE on success

rduckhts_gff *Create GFF3 Table*

Description

Creates a DuckDB table from GFF3 files using the DuckHTS extension.

Usage

```
rduckhts_gff(
  con,
  table_name,
  path,
  region = NULL,
  attributes_map = FALSE,
  overwrite = FALSE
)
```

Arguments

con	A DuckDB connection with DuckHTS loaded
table_name	Name for the created table
path	Path to the GFF3 file
region	Optional genomic region (e.g., "chr1:1000-2000")
attributes_map	Logical. If TRUE, returns attributes as a MAP column
overwrite	Logical. If TRUE, overwrites existing table

Value

Invisible TRUE on success

---

rduckhts_gtf *Create GTF Table*

Description

Creates a DuckDB table from GTF files using the DuckHTS extension.

Usage

```
rduckhts_gtf(
  con,
  table_name,
  path,
  region = NULL,
  attributes_map = FALSE,
  overwrite = FALSE
)
```

Arguments

con	A DuckDB connection with DuckHTS loaded
table_name	Name for the created table
path	Path to the GTF file
region	Optional genomic region (e.g., "chr1:1000-2000")
attributes_map	Logical. If TRUE, returns attributes as a MAP column
overwrite	Logical. If TRUE, overwrites existing table

Value

Invisible TRUE on success

---

rduckhts_load *Load DuckHTS Extension*

Description

Loads the DuckHTS extension into a DuckDB connection. This must be called before using any of the HTS reader functions.

Usage

```
rduckhts_load(con, extension_path = NULL)
```

Arguments

con	A DuckDB connection object
extension_path	Optional path to the duckhts extension file. If NULL, will try to use the bundled extension.

Details

The DuckDB connection must be created with `allow_unsigned_extensions = "true"`.

Value

TRUE if the extension was loaded successfully

Examples

```
library(DBI)
library(duckdb)

con <- dbConnect(duckdb::duckdb(config = list(allow_unsigned_extensions = "true")))
rduckhts_load(con)
dbDisconnect(con, shutdown = TRUE)
```

`rduckhts_tabix` *Create Tabix-Indexed File Table*

Description

Creates a DuckDB table from any tabix-indexed file using the DuckHTS extension.

Usage

```
rduckhts_tabix(
  con,
  table_name,
  path,
  region = NULL,
  header = NULL,
  header_names = NULL,
  auto_detect = NULL,
  column_types = NULL,
  overwrite = FALSE
)
```

Arguments

`con`	A DuckDB connection with DuckHTS loaded
`table_name`	Name for the created table
`path`	Path to the tabix-indexed file
`region`	Optional genomic region (e.g., "chr1:1000-2000")
`header`	Logical. If TRUE, use first non-meta line as column names
`header_names`	Character vector to override column names
`auto_detect`	Logical. If TRUE, infer basic numeric column types
`column_types`	Character vector of column types (e.g. "BIGINT", "VARCHAR")
`overwrite`	Logical. If TRUE, overwrites existing table

Value

Invisible TRUE on success

setup_hts_env *Setup HTSlib Environment*

Description

Sets the 'HTS_PATH' environment variable to point to the bundled htslib plugins directory. This enables remote file access via libcurl plugins (e.g., s3://, gs://, http://) when plugins are available.

Usage

```
setup_hts_env(plugins_dir = NULL)
```

Arguments

plugins_dir	Optional path to the htslib plugins directory. When NULL, uses the bundled plugins directory if available.

Details

Call this before querying remote URLs to allow htslib to locate its plugins.

Value

Invisibly returns the previous value of 'HTS_PATH' (or 'NA' if unset).

Examples

```
## Not run:
setup_hts_env()

plugins_path <- tempfile("hts_plugins_")
dir.create(plugins_path)
setup_hts_env(plugins_dir = plugins_path)
unlink(plugins_path, recursive = TRUE)

## End(Not run)
```

Index