

# Package: ClickHouseHTTP (via r-universe)

September 4, 2024

**Type** Package

**Title** A Simple HTTP Database Interface to 'ClickHouse'

**Version** 0.3.3

**Description** 'ClickHouse' (<<https://clickhouse.com/>>) is an open-source, high performance columnar OLAP (online analytical processing of queries) database management system for real-time analytics using SQL. This 'DBI' backend relies on the 'ClickHouse' HTTP interface and support HTTPS protocol.

**URL** <https://github.com/patzaw/ClickHouseHTTP>

**BugReports** <https://github.com/patzaw/ClickHouseHTTP/issues>

**Depends** R (>= 3.6)

**Imports** methods, DBI (>= 0.3.0), httr, jsonlite, arrow, data.table

**Suggests** knitr, rmarkdown, dplyr, stringi

**Roxygen** list(markdown = TRUE)

**License** GPL-3

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**Repository** <https://patzaw.r-universe.dev>

**RemoteUrl** <https://github.com/patzaw/clickhousehttp>

**RemoteRef** HEAD

**RemoteSha** 6d84e39c8b5d229de5f094133d040a9a7abcc1ef

## Contents

ClickHouseHTTP . . . . .	2
ClickHouseHTTPConnection-class . . . . .	2
ClickHouseHTTPDriver-class . . . . .	9
ClickHouseHTTPResult-class . . . . .	11

<b>Index</b>	<b>12</b>
--------------	-----------

---

ClickHouseHTTP      *Create a ClickHouseHTTP DBI driver*

---

**Description**

Create a ClickHouseHTTP DBI driver

**Usage**

ClickHouseHTTP()

**Value**

A ClickHouseHTTPDriver

**See Also**

[ClickHouseHTTPDriver](#)

---

ClickHouseHTTPConnection-class  
*ClickHouseHTTPConnection class.*

---

**Description**

ClickHouseHTTPConnection class.  
 Send SQL query to ClickHouse  
 Information about the ClickHouse database  
 Create a table in ClickHouse  
 Write a table in ClickHouse

**Usage**

```
## S4 method for signature 'ClickHouseHTTPConnection,character'
dbSendQuery(
  conn,
  statement,
  format = c("Arrow", "TabSeparatedWithNamesAndTypes"),
  file = NA,
  ...
)

## S4 method for signature 'ClickHouseHTTPConnection'
dbGetInfo(dbObj, ...)
```

```

## S4 method for signature 'ClickHouseHTTPConnection'
dbCreateTable(
  conn,
  name,
  fields,
  engine = "TinyLog",
  overwrite = FALSE,
  ...,
  row.names = NULL,
  temporary = FALSE
)

## S4 method for signature 'ClickHouseHTTPConnection,ANY'
dbWriteTable(
  conn,
  name,
  value,
  overwrite = FALSE,
  append = FALSE,
  engine = "TinyLog",
  ...
)

```

### Arguments

conn	a ClickHouseHTTPConnection object created with <a href="#">dbConnect()</a>
statement	the SQL query statement
format	the format used by ClickHouse to send the results. Two formats are supported: "Arrow" (default) and "TabSeparatedWithNamesAndTypes"
file	a path to a file to send along the query (default: NA)
...	Other parameters passed on to methods
dbObj	a ClickHouseHTTPConnection object
name	the name of the table to create
fields	a character vector with the name of the fields and their ClickHouse type (e.g. <code>c("text_col String", "num_col Nullable(Float64)", "nul_col Array(Int32)")</code> )
engine	the ClickHouse table engine as described in <a href="#">ClickHouse documentation</a> . Examples: <ul style="list-style-type: none"> <li>"TinyLog" (default)</li> <li>"MergeTree() ORDER BY (expr)" (expr generally correspond to fields separated by ",")</li> </ul>
overwrite	if TRUE and if a table with the same name exists, then it is deleted before creating the new one (default: FALSE)
row.names	unsupported parameter (add for compatibility reason)

temporary	unsupported parameter (add for compatibility reason)
value	the table to write
append	if TRUE, the values are added to the database table if it exists (default: FALSE).

### Details

Both format have their pros and cons:

- **Arrow** (default):
  - fast for long tables but slow for wide tables
  - fast with Array columns
  - Date and DateTime columns are returned as UInt16 and UInt32 respectively: by default, ClickHouseHTTP interpret them as Date and POSIXct columns but cannot make the difference with actual UInt16 and UInt32
- **TabSeparatedWithNamesAndTypes**:
  - in general faster than Arrow
  - fast for wide tables but slow for long tables
  - slow with Array columns
  - Special characters are not well interpreted. In such cases, the function below can be useful but can also take time.

```
.sp_ch_recov <- function(x){
  stringi::stri_replace_all_regex(
    x,
    c(
      "\\n", "\\t", "\\r", "\\b",
      "\\a", "\\f", "\\'", "\\\"\\\"
    ),
    c("\\n", "\\t", "\\r", "\\b", "\\a", "\\f", "'", "\\\""),
    vectorize_all=FALSE
  )
}
```

### Value

A ClickHouseHTTPResult object

A list with the following elements:

- name: "ClickHouseHTTPConnection"
- db.version: the version of ClickHouse
- uptime: ClickHouse uptime
- dbname: the default database
- username: user name
- host: ClickHouse host
- port: ClickHouse port

- `https`: Is the connection using HTTPS protocol instead of HTTP

`dbCreateTable()` returns TRUE, invisibly.

TRUE; called for side effects

## See Also

[ClickHouseHTTPResult](#)

## Examples

```
## Not run:

## Connection ----

library(DBI)
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8123
)

### HTTPS connection (without ssl peer verification) ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8443, https=TRUE, ssl_verifypeer=FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames="car")
dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
  con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format="TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "type")
```

```

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
data("swiss")
swiss <- as_tibble(swiss, rownames="province")
swiss <- mutate(swiss, "pr letters"=strsplit(province, ""))
dbWriteTable(
  con, "swiss", swiss,
  engine="MergeTree() ORDER BY (Fertility, province)"
)
swissFromDB <- dbReadTable(con, "swiss")

## A table from another database can also be accessed as following:
dbReadTable(con, SQL("default.mtcars"))

## End(Not run)
## Not run:

## Connection ----

library(DBI)
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8123
)

### HTTPS connection (without ssl peer verification) ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8443, https=TRUE, ssl_verifypeer=FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames="car")
dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

```

```
carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
  con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format="TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "type")

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
data("swiss")
swiss <- as_tibble(swiss, rownames="province")
swiss <- mutate(swiss, "pr letters"=strsplit(province, ""))
dbWriteTable(
  con, "swiss", swiss,
  engine="MergeTree() ORDER BY (Fertility, province)"
)
swissFromDB <- dbReadTable(con, "swiss")

## A table from another database can also be accessed as following:
dbReadTable(con, SQL("default.mtcars"))

## End(Not run)
## Not run:

## Connection ----

library(DBI)
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8123
)

### HTTPS connection (without ssl peer verification) ----
```

```

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8443, https=TRUE, ssl_verifypeer=FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames="car")
dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
  con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format="TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "type")

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
data("swiss")
swiss <- as_tibble(swiss, rownames="province")
swiss <- mutate(swiss, "pr letters"=strsplit(province, ""))
dbWriteTable(
  con, "swiss", swiss,
  engine="MergeTree() ORDER BY (Fertility, province)"
)
swissFromDB <- dbReadTable(con, "swiss")

## A table from another database can also be accessed as following:
dbReadTable(con, SQL("default.mtcars"))

```



```
## End(Not run)
```

---

```
ClickHouseHTTPDriver-class
```

*Driver for the ClickHouse database using HTTP(S) interface*

---

## Description

Driver for the ClickHouse database using HTTP(S) interface

Connect to a ClickHouse database using the ClickHouseHTTP DBI

## Usage

```
## S4 method for signature 'ClickHouseHTTPDriver'
dbConnect(
  drv,
  host = "localhost",
  port = 8123L,
  dbname = "default",
  user = "default",
  password = "",
  https = FALSE,
  ssl_verifypeer = TRUE,
  host_path = NA,
  session_timeout = 3600L,
  convert_uint = TRUE,
  extended_headers = list(),
  reset_handle = FALSE,
  ...
)
```

## Arguments

<code>drv</code>	A driver object created by <code>ClickHouseHTTP()</code>
<code>host</code>	name of the database host (default: "localhost")
<code>port</code>	port on which the database is listening (default: 8123L)
<code>dbname</code>	name of the default database (default: "default")
<code>user</code>	user name (default: "default")
<code>password</code>	user password (default: "")
<code>https</code>	a logical to use the HTTPS protocol (default: FALSE)
<code>ssl_verifypeer</code>	a logical to verify SSL certificate when using HTTPS (default: TRUE)
<code>host_path</code>	a path to use on host (e.g. "ClickHouse/"): it allows to connect on a server behind a reverse proxy for example

**session\_timeout** timeout in seconds (default: 3600L seconds)  
**convert\_uint** a logical: if TRUE (default), UInt ClickHouse data types are converted in the following R classes:
 

- UInt8 : logical
- UInt16: Date (when using Arrow format in [dbSendQuery, ClickHouseHTTPConnection, character-method](#))
- UInt32: POSIXct (when using Arrow format in [dbSendQuery, ClickHouseHTTPConnection, character-method](#))

**extended\_headers** a named list with other HTTP headers (for example: `extended_headers=list("X-Authorization"="Bearer <token>")` can be used for OAuth access delegation)  
**reset\_handle** a logical indicating how to manage Curl handles (see [httr::handle\\_pool](#)). If TRUE, handle reset is used (default: FALSE).  
**...** Other parameters passed on to methods

**Value**

A ClickHouseHTTPConnection

**See Also**

[ClickHouseHTTPConnection](#)

**Examples**

```

## Not run:

## Connection ----

library(DBI)
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8123
)

### HTTPS connection (without ssl peer verification) ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8443, https=TRUE, ssl_verifypeer=FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames="car")

```

```

dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
  con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format="TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "type")

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
data("swiss")
swiss <- as_tibble(swiss, rownames="province")
swiss <- mutate(swiss, "pr letters"=strsplit(province, ""))
dbWriteTable(
  con, "swiss", swiss,
  engine="MergeTree() ORDER BY (Fertility, province)"
)
swissFromDB <- dbReadTable(con, "swiss")

## A table from another database can also be accessed as following:
dbReadTable(con, SQL("default.mtcars"))

## End(Not run)

```

---

ClickHouseHTTPResult-class

*ClickHouseHTTPResult class.*


---

## Description

ClickHouseHTTPResult class.

# Index

ClickHouseHTTP, [2](#)  
ClickHouseHTTP(), [9](#)  
ClickHouseHTTPConnection, [10](#)  
ClickHouseHTTPConnection-class, [2](#)  
ClickHouseHTTPDriver, [2](#)  
ClickHouseHTTPDriver-class, [9](#)  
ClickHouseHTTPResult, [5](#)  
ClickHouseHTTPResult-class, [11](#)

dbConnect(), [3](#)  
dbConnect, ClickHouseHTTPDriver-method  
(ClickHouseHTTPDriver-class), [9](#)  
dbCreateTable, ClickHouseHTTPConnection-method  
(ClickHouseHTTPConnection-class),  
[2](#)  
dbGetInfo, ClickHouseHTTPConnection-method  
(ClickHouseHTTPConnection-class),  
[2](#)  
dbSendQuery, ClickHouseHTTPConnection, character-method,  
[10](#)  
dbSendQuery, ClickHouseHTTPConnection, character-method  
(ClickHouseHTTPConnection-class),  
[2](#)  
dbWriteTable, ClickHouseHTTPConnection, ANY-method  
(ClickHouseHTTPConnection-class),  
[2](#)

htr::handle\_pool, [10](#)