

# Package ‘ghql’

May 8, 2026

**Type** Package

**Title** General Purpose 'GraphQL' Client

**Version** 0.1.2

**Description** A 'GraphQL' client, with an R6 interface for initializing a connection to a 'GraphQL' instance, and methods for constructing queries, including fragments and parameterized queries. Queries are checked with the 'libgraphqlparser' C++ parser via the 'graphql' package.

**License** MIT + file LICENSE

**URL** <https://docs.ropensci.org/ghql/>, <https://github.com/ropensci/ghql>

**BugReports** <https://github.com/ropensci/ghql/issues>

**Imports** crul, graphql, jsonlite, R6

**Suggests** testthat

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.3.2

**X-schema.org-applicationCategory** Web

**X-schema.org-isPartOf** <https://ropensci.org>

**X-schema.org-keywords** http, API, web-services, curl, data, GraphQL

**NeedsCompilation** no

**Author** Scott Chamberlain [aut] (ORCID:  
<<https://orcid.org/0000-0003-1444-9135>>),  
Mark Padgham [aut, cre] (ORCID:  
<<https://orcid.org/0000-0003-2172-5265>>)

**Maintainer** Mark Padgham <[mark.padgham@email.com](mailto:mark.padgham@email.com)>

**Repository** CRAN

**Date/Publication** 2025-09-02 22:00:02 UTC

## Contents

ghql-package . . . . .	2
Fragment . . . . .	3
GraphqlClient . . . . .	4
Query . . . . .	9
<b>Index</b>	<b>13</b>

---

ghql-package	<i>ghql</i>
--------------	-------------

---

## Description

General purpose GraphQL client

## ghql API

The main interface in this package is [GraphqlClient], which produces a client (R6 class) with various methods for interacting with a GraphQL server. [GraphqlClient] also accepts various input parameters to set a base URL, and any headers required, which is usually the required set of things needed to connect to a GraphQL service.

[Query] is an interface to creating GraphQL queries, which works together with [GraphqlClient]

[Fragment] is an interface to creating GraphQL fragments, which works together with [GraphqlClient]

## Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

## See Also

Useful links:

- <https://docs.ropensci.org/ghql/>
- <https://github.com/ropensci/ghql>
- Report bugs at <https://github.com/ropensci/ghql/issues>

---

Fragment

*Fragment*

---

## Description

ghql fragment class

## Value

a 'Fragment' class (R6 class)

## Public fields

fragments (list) list of fragments

## Methods

### Public methods:

- [Fragment\\$print\(\)](#)
- [Fragment\\$fragment\(\)](#)

**Method** `print()`: print method for the 'Fragment' class

*Usage:*

```
Fragment$print(x, ...)
```

*Arguments:*

x self

... ignored

**Method** `fragment()`: create a fragment by name

*Usage:*

```
Fragment$fragment(name, x)
```

*Arguments:*

name (character) fragment name

x (character) the fragment

*Returns:* nothing returned; sets fragments internally

## Examples

```
# make a fragment class
frag <- Fragment$new()

# define a fragment
frag$fragment('Watchers', '
  fragment on Repository {
    watchers(first: 3) {
```

```

        edges {
          node {
            name
          }
        }
      }
    }')

# define another fragment
frag$fragment('Stargazers', '
  fragment on Repository {
    stargazers(first: 3) {
      edges {
        node {
          name
        }
      }
    }
  }')
frag
frag$fragments
frag$fragments$Watchers
frag$fragments$Stargazers

```

---

GraphQLClient

*GraphQLClient*


---

## Description

R6 class for constructing GraphQL queries

## Value

a 'GraphQLClient' class (R6 class)

## Public fields

url (character) list of fragments  
 headers list of named headers  
 schema holds schema  
 result holds result from http request  
 fragments (list) list of fragments

## Methods

### Public methods:

- [GraphQLClient\\$new\(\)](#)

- GraphQLClient#print()
- GraphQLClient\$ping()
- GraphQLClient\$load\_schema()
- GraphQLClient\$dump\_schema()
- GraphQLClient\$schema2json()
- GraphQLClient\$fragment()
- GraphQLClient\$exec()
- GraphQLClient\$prep\_query()

**Method** new(): Create a new 'GraphQLClient' object

*Usage:*

GraphQLClient\$new(url, headers)

*Arguments:*

url (character) URL for the GraphQL schema

headers Any acceptable headers, a named list. See examples

*Returns:* A new 'GraphQLClient' object

**Method** print(): print method for the 'GraphQLClient' class

*Usage:*

GraphQLClient#print(x, ...)

*Arguments:*

x self

... ignored

**Method** ping(): ping the GraphQL server

*Usage:*

GraphQLClient\$ping(...)

*Arguments:*

... curl options passed on to [curl::verb-HEAD]

*Returns:* 'TRUE' if successful response, 'FALSE' otherwise

**Method** load\_schema(): load schema, from URL or local file

*Usage:*

GraphQLClient\$load\_schema(schema\_url = NULL, schema\_file = NULL, ...)

*Arguments:*

schema\_url (character) url for a schema file

schema\_file (character) path to a schema file

... curl options passed on to [curl::verb-GET]

*Returns:* nothing, loads schema into '\$schema' slot

**Method** dump\_schema(): dump schema to a local file

*Usage:*

GraphQLClient\$dump\_schema(file)

*Arguments:*

file (character) path to a file

*Returns:* nothing, writes schema to 'file'

**Method** schema2json(): convert schema to JSON

*Usage:*

GraphQLClient\$schema2json(...)

*Arguments:*

... options passed on to [jsonlite::toJSON()]

*Returns:* json

**Method** fragment(): load schema, from URL or local file

*Usage:*

GraphQLClient\$fragment(name, x)

*Arguments:*

name (character) fragment name

x (character) the fragment

*Returns:* nothing returned; sets fragments internally

**Method** exec(): execute the query

*Usage:*

```
GraphQLClient$exec(
  query,
  variables,
  encoding = "UTF-8",
  response_headers = FALSE,
  ...
)
```

*Arguments:*

query (character) a query, of class 'query' or 'fragment'

variables (list) named list with query variables values

encoding (character) encoding to use to parse the response. passed on to [crul::HttpResponse]\$parse() method. default: "UTF-8"

response\_headers If 'TRUE', include the response headers as an attribute of the return object.

... curl options passed on to [crul::verb-POST]

*Returns:* character string of response, if successful

**Method** prep\_query(): not used right now

*Usage:*

GraphQLClient\$prep\_query(query)

*Arguments:*

query (character) a query, of class 'query' or 'fragment'

**Examples**

```
x <- GraphQLClient$new()
x

## Not run:
# make a client
token <- Sys.getenv("GITHUB_TOKEN")
cli <- GraphQLClient$new(
  url = "https://api.github.com/graphql",
  headers = list(Authorization = paste0("Bearer ", token))
)

# if the GraphQL server has a schema, you can load it
cli$load_schema()

# dump schema to local file
f <- tempfile(fileext = ".json")
cli$dump_schema(file = f)
readLines(f)
jsonlite::fromJSON(readLines(f))

# after dumping to file, you can later read schema from file for faster loading
rm(cli)
cli <- GraphQLClient$new(
  url = "https://api.github.com/graphql",
  headers = list(Authorization = paste0("Bearer ", token))
)
cli$load_schema(schema_file = f)

# variables
cli$url
cli$schema
cli$schema$data
cli$schema$data$`__schema`
cli$schema$data$`__schema`$queryType
cli$schema$data$`__schema`$mutationType
cli$schema$data$`__schema`$subscriptionType
head(cli$schema$data$`__schema`$types)
cli$schema$data$`__schema`$directives

# methods
## ping - hopefully you get TRUE
cli$ping()

## dump schema
cli$schema2json()

## define query
### creat a query class first
qry <- Query$new()
```

```
## another
qry$query('repos', '{
  viewer {
    repositories(last: 10, isFork: false, privacy: PUBLIC) {
      edges {
        node {
          isPrivate
          id
          name
        }
      }
    }
  }
}')
qry
qry$queries
qry$queries$repos
### execute the query
cli$exec(qry$queries$repos)
```

```
# query with a fragment
### define query without fragment, but referring to it
qry <- Query$new()
qry$query('queryfrag', '{
  ropensci: repositoryOwner(login:"ropensci") {
    repositories(first: 3) {
      edges {
        node {
          ...Watchers
        }
      }
    }
  }
  ropenscilabs: repositoryOwner(login:"ropenscilabs") {
    repositories(first: 3) {
      edges {
        node {
          ...Watchers
        }
      }
    }
  }
}')
}
```

```
### define a fragment
frag <- Fragment$new()
frag$fragment('Watchers', '
  fragment on Repository {
    watchers(first: 3) {
      edges {
        node {
          name
        }
      }
    }
  }
}')
}
```

```

    }
  }
}
}')
frag$fragments
frag$fragments$Watchers

### add the fragment to the query 'queryfrag'
qry$add_fragment('queryfrag', frag$fragments$Watchers)
qry
qry$queries$queryfrag

### execute query: we'll hook together the query and your fragment internally
cli$exec(qry$queries$queryfrag)

## End(Not run)

```

---

Query

*Query*


---

## Description

ghql query class

## Value

a 'Query' class (R6 class)

## Public fields

queries (list) list of queries

## Methods

### Public methods:

- [Query#print\(\)](#)
- [Query\\$query\(\)](#)
- [Query\\$add\\_fragment\(\)](#)
- [Query\\$parse2json\(\)](#)

**Method print():** print method for the 'Query' class

*Usage:*

Query#print(x, ...)

*Arguments:*

x self

... ignored

**Method** `query()`: define query in a character string

*Usage:*

```
Query$query(name, x)
```

*Arguments:*

name (character) name of the query

x (character) the query

*Returns:* nothing returned; sets query with 'name' internally

**Method** `add_fragment()`: add a fragment to a query

*Usage:*

```
Query$add_fragment(query_name, fragment)
```

*Arguments:*

query\_name (character) the query name to add the fragment to

fragment (character) the fragment itself

*Returns:* nothing returned; sets the fragment with the query

**Method** `parse2json()`: parse query string with libgraphqlparser and get back JSON

*Usage:*

```
Query$parse2json(query, parse_schema = FALSE)
```

*Arguments:*

query (character) a query to parse

parse\_schema (logical) enable schema definition parsing? default: 'FALSE'

*Returns:* adf

## Note

we run an internal method 'check\_query()' that runs the public method 'parse2json()' - if the query doesn't pass the libgraphqlparser parser, we return the error message

## Examples

```
# make a client
qry <- Query$new()

## define query
qry$query('query2', '{
  viewer {
    repositories(last: 10, isFork: false, privacy: PUBLIC) {
      edges {
        node {
          isPrivate
          id
          name
        }
      }
    }
  }
}
```

```

    }
  })
  qry
  qry$queries
  qry$queries$query2

# fragments
## by hand
qry$query('querywithfrag', '{
  ropensci: repositoryOwner(login:"ropensci") {
    repositories(first: 3) {
      edges {
        node {
          ...Watchers
        }
      }
    }
  }
  ropenscilabs: repositoryOwner(login:"ropenscilabs") {
    repositories(first: 3) {
      edges {
        node {
          ...Watchers
        }
      }
    }
  }
}')
qry
qry$queries
qry$queries$querywithfrag

fragment Watchers on Repository {
  watchers(first: 3) {
    edges {
      node {
        name
      }
    }
  }
}')
qry
qry$queries
qry$queries$querywithfrag

## Not run:
token <- Sys.getenv("GITHUB_TOKEN")
con <- GraphQLClient$new(
  url = "https://api.github.com/graphql",
  headers = list(Authorization = paste0("Bearer ", token))
)
jsonlite::fromJSON(con$exec(qry$queries$querywithfrag))

## use Fragment class fragments generator
### define query without fragment, but referring to it
qry$query('queryfrag', '{
```

```

ropensci: repositoryOwner(login:"ropensci") {
  repositories(first: 3) {
    edges {
      node {
        ...Watchers
      }
    }
  }
}
ropenscilabs: repositoryOwner(login:"ropenscilabs") {
  repositories(first: 3) {
    edges {
      node {
        ...Watchers
      }
    }
  }
}
}')

```

### define a fragment, and use it later

```

frag <- Fragment$new()
frag$fragment('Watchers', '
  fragment on Repository {
    watchers(first: 3) {
      edges {
        node {
          name
        }
      }
    }
  }
}')
frag$fragments
frag$fragments$Watchers

```

### add the fragment to the query 'queryfrag'

```

qry$add_fragment('queryfrag', frag$fragments$Watchers)
qry
qry$queries
qry$queries$queryfrag

```

## End(Not run)

# Index

\* **package**

ghql-package, [2](#)

Fragment, [3](#)

ghql (ghql-package), [2](#)

ghql-package, [2](#)

GraphQLClient, [4](#)

Query, [9](#)