Package 'virtuoso'

September 1, 2020

Type Package

Title Interface to 'Virtuoso' using 'ODBC'

Version 0.1.5

Description Provides users with a simple and convenient
mechanism to manage and query a 'Virtuoso' database using the 'DBI' (Data-Base Interface)
compatible 'ODBC' (Open Database Connectivity) interface.
'Virtuoso' is a high-performance ``universal server,'' which can act
as both a relational database, supporting standard Structured Query
Language ('SQL') queries, while also supporting data following the
Resource Description Framework ('RDF') model for Linked Data.
'RDF' data can be queried using 'SPARQL' ('SPARQL' Protocol and 'RDF' Query Language)
queries, a graph-based query that supports semantic reasoning.
This allows users to leverage the performance of local or remote 'Virtuoso' servers using
popular 'R' packages such as 'DBI' and 'dplyr', while also providing a
high-performance solution for working with large 'RDF' 'triplestores' from 'R.'
The package also provides helper routines to install, launch, and manage
a 'Virtuoso' server locally on 'Mac', 'Windows' and 'Linux' platforms using
the standard interactive installers from the 'R' command-line. By
automatically handling these setup steps, the package can make using 'Virtuoso'
considerably faster and easier for a most users to deploy in a local
environment. Managing the bulk import of triples
from common serializations with a single intuitive command is another key
feature of this package. Bulk import performance can be tens to
hundreds of times faster than the comparable imports using existing 'R' tools,
including 'rdflib' and 'redland' packages.

License MIT + file LICENSE

URL https://github.com/ropensci/virtuoso

BugReports https://github.com/ropensci/virtuoso/issues

Encoding UTF-8

LazyData true

Imports odbc, processx, DBI, utils, ini, rappdirs, curl, fs, digest,
ps

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, nycflights13, testthat, covr, jsonld, dplyr, spelling

VignetteBuilder knitr

Language en-US

SystemRequirements virtuoso-opensource (Linux). For Mac & Windows, this package can automate Virtuoso installation.

NeedsCompilation no

Author Carl Boettiger [aut, cre, cph] (<https://orcid.org/0000-0002-1642-628X>), Bryce Mecum [ctb] (<https://orcid.org/0000-0002-0381-3766>)

Maintainer Carl Boettiger <cboettig@gmail.com>

Repository CRAN

Date/Publication 2020-09-01 08:40:03 UTC

R topics documented:

`has_virtuoso` *check for Virtuoso*

Description

test if the system has a virtuoso installation on the path

Usage

```
has_virtuoso()
```

Value

logical indicating if virtuoso-t binary was found or now.

Examples

```
has_virtuoso()
```

---

vos_configure *Configure Virtuoso Server ini file*

Description

Virtuoso Server configuration is determined by a virtuoso.ini file when server starts. This file includes both system-specific information from your install (location of server files, addons, etc) and user-configurable parameters. This helper function provides a way to create and modify an appropriate `virtuoso.ini` file.

Usage

```
vos_configure(
  dirs_allowed = getwd(),
  gigs_ram = 2,
  template = find_virtuoso_ini(),
  db_dir = vos_db()
)
```

Arguments

`dirs_allowed`	Paths (relative or absolute) to directories from which Virtuoso should have read and write access (e.g. for bulk uploading). Should be specified as a single comma-separated string.
`gigs_ram`	Indicate approximately the maximum GB of memory Virtuoso can have access to. (Used to set NumberOfBuffers & MaxDirtyBuffers in config.)
`template`	Location of an existing virtuoso.ini file which will be used as a template. By default, `vos_configure()` will attempt to locate the appropriate template for your system.
`db_dir`	location where `virtuoso.ini` file should be written. Other Virtuoso database log files will also be written here.

Value

Writes the requested `virtuoso.ini` file to the db_dir specified and returns the path to this file.

References

http://docs.openlinksw.com/virtuoso/dbadm/

Examples

```
# can take > 5s to test
## configure with typical defaults:
vos_configure()
## Increase or decrease RAM available to virtuoso:
vos_configure(gigs_ram = 1)
```

`vos_connect` *Connect to a Virtuoso Server over ODBC*

Description

Connect to a Virtuoso Server over ODBC

Usage

```
vos_connect(
  driver = NULL,
  uid = "dba",
  pwd = "dba",
  host = "localhost",
  port = "1111",
  system_odbcinst = find_odbcinst(),
  local_odbcinst = odbcinst_path()
)
```

Arguments

`driver`	Name of the Driver line in the ODBC configuration
`uid`	User id. Defaults to "dba"
`pwd`	Password. Defaults to "dba"
`host`	IP address of the Virtuoso Server
`port`	Port used by Virtuoso. Defaults to the Virtuoso standard port, 1111
`system_odbcinst`	Path to the system `odbcinst.ini` file. (Does not require write access.) Default will attempt to find the file for your system.
`local_odbcinst`	Path to the local odbcinst we should use.

Details

Default parameters are appropriate for the automatic installer provided by the package and for the default settings typically used by local Virtuoso installers. Adjust these only if you are connecting to a remote virtuoso server that is not controlled from the R package.

Value

a DBI connection to the Virtuoso database. This can be passed to additional virtuoso functions such as `vos_import()` or `vos_query()`, and can also be used as a standard DBI or dplyr database backend.

See Also

`vos_install()`, `vos_start()`

Examples

```
vos_status()

if(has_virtuoso()){
## start up
vos_start()
con <- vos_connect()
}
```

---

vos_delete_db *Delete Virtuoso Database*

Description

delete the entire Virtuoso database for a fresh start.

Usage

```
vos_delete_db(ask = is_interactive(), db_dir = vos_db())
```

Arguments

ask	ask before deleting?
db_dir	location of the directory to delete

Examples

```
vos_delete_db()
```

vos_destroy_all *Destroy all Virtuoso's directories*

Description

Provides a clean reset of the system that purges all data files, config files, cache and log files created by virtuoso R package. This does not uninstall Virtuoso software itself, see `vos_uninstall()` to uninstall.

Usage

```
vos_destroy_all(force = FALSE)
```

Arguments

`force`	should permissions be changed (if possible) to allow deletion?

Value

TRUE if entirely successful in removing all files, FALSE otherwise (invisibly).

Examples

```
vos_destroy_all()
```

vos_import *Bulk Import of RDF triples*

Description

While triples data can be added one by one over SPARQL queries, Virtuoso bulk import is by far the fastest way to import large triplestores in the database.

Usage

```
vos_import(
  con,
  files = NULL,
  wd = ".",
  glob = "*",
  graph = "rdflib",
  n_cores = 1L
)
```

Arguments

con	a ODBC connection to Virtuoso, from vos_connect()
files	paths to files to be imported
wd	Alternatively, can specify directory and globbing pattern to import. Note that in this case, wd must be in (or a subdir of) the AllowedDirs list of virtuoso.ini file created by vos_configure(). By default, this includes the working directory where you called vos_start() or vos_configure().
glob	A wildcard aka globbing pattern (e.g. ‘"*.nq"‘).
graph	Name (technically URI) for a graph in the database. Can leave as default. If a graph is already specified by the import file (e.g. in nquads), that will be used instead.
n_cores	specify the number of available cores for parallel loading. Particularly useful when importing large numbers of bulk files.

Details

the bulk importer imports all files matching a pattern in a given directory. If given a list of files, these are temporarily symlinked (or copied on Windows machines) to the Virtuoso app cache dir in a subdirectory, and the entire subdirectory is loaded (filtered by the globbing pattern). If files are not specified, load is called directly on the specified directory and pattern. This is particularly useful for loading large numbers of files.

Note that Virtuoso recommends breaking large files into multiple smaller ones, which can improve loading time (particularly if using multiple cores.)

Virtuoso Bulk Importer recognizes the following file formats:

- .grdf
- .nq
- .owl
- .nt
- .rdf
- .trig
- .ttl
- .xml

Any of these can optionally be gzipped (with a .gz extension).

Value

(Invisibly) returns the status table of the bulk loader, indicating file loading time or errors.

References

http://vos.openlinksw.com/owiki/wiki/VOS/VirtBulkRDFLoader

Examples

```
vos_status()


if(has_virtuoso()){
vos_start()
con <- vos_connect()

example <- system.file("extdata", "person.nq", package = "virtuoso")
vos_import(con, example)
}
```

`vos_install` *Helper method for installing Virtuoso Server*

Description

Installation helper for Mac and Windows machines. By default, method will download and launch the official `.dmg` or `.exe` installer for your platform, running the standard drag-n-drop installer or interactive dialog. Setting `ask = FALSE` will allow the installer to run entirely unsupervised, which is suitable for use in scripts. Mac users can alternatively opt to install Virtuoso through HomeBrew by setting `use_brew=TRUE`. Linux users should simply install the `virtuoso-opensource` package (e.g. in debian & ubuntu) using the package manager or by contacting your system administrator.

Usage

```
vos_install(ask = is_interactive(), use_brew = FALSE)
```

Arguments

`ask`	Should we ask user for interactive installation?
`use_brew`	Should we use homebrew to install? (MacOS only)

See Also

`vos_start()`, `vos_uninstall()`

Examples

```
vos_install()
```

vos_kill *Stop (kill) the Virtuoso server*

Description

Kill ends the process started by `vos_start()`

Usage

```
vos_kill(p = NA)
```

Arguments

p	a process object, returned by `vos_process()` or `vos_start()`. (will be restored from cache if not provided)

Details

vos_kill simply shuts down the local Virtuoso server, it does not remove any data stored in the database system. `vos_kill()` terminates the process, removing the process id from the process table.

See Also

`vos_start()`

Examples

```
if(has_virtuoso()){

  vos_start()
  vos_kill()

  }
```

vos_list_graphs *List graphs*

Description

List graphs

Usage

```
vos_list_graphs(con)
```

Arguments

con	a ODBC connection to Virtuoso, from vos_connect()

Examples

```
vos_status()

if(has_virtuoso()){
vos_start()
con <- vos_connect()
vos_list_graphs(con)

}
```

---

vos_log *Query the server logs*

Description

Query the server logs

Usage

```
vos_log(p = NA, collapse = NULL, just_errors = FALSE)
```

Arguments

p	a process object, returned by vos_process() or vos_start(). (will be restored from cache if not provided)
collapse	an optional character string to separate the lines in a single character string.
just_errors	logical, default FALSE. Set to TRUE to return just the lines that contain the term "error", which can be useful in debugging or validating bulk imports.

Value

Virtuoso logs as a character vector.

See Also

vos_start()

Examples

```
if(has_virtuoso())
  vos_log()
```

vos_odbcinst *Configure the ODBC Driver for Virtuoso*

Description

ODBC uses an `odbcinst.ini` file to point ODBC at the library required to drive any given database. This function helps us automatically locate the driver library on different operating systems and configure the odbcinst appropriately for each OS.

Usage

```
vos_odbcinst(
  system_odbcinst = find_odbcinst(),
  local_odbcinst = odbcinst_path()
)
```

Arguments

`system_odbcinst`	Path to the system `odbcinst.ini` file. (Does not require write access.) Default will attempt to find the file for your system.
`local_odbcinst`	Path to the local odbcinst we should use.

Details

This function is called automatically by `vos_install()` and thus does not usually need to be called by the user. Users can also manually configure ODBC as outlined in https://github.com/r-dbi/odbc#dsn-configuration-files. This is merely a convenience function automating that process on most systems.

Value

the path to the odbcinst file that is created or modified.

Examples

```
## Configures ODBC and returns silently on success.
vos_odbcinst()

## see where the inst file is located:
inst <- vos_odbcinst()
inst
```

vos_process *Return a handle to an existing Virtuoso Process*

Description

Generally a user will not need to access this function directly, though it may be useful for debugging purposes.

Usage

```
vos_process(p = NA)
```

Arguments

p	a process object, returned by vos_process() or vos_start(). (will be restored from cache if not provided)

Value

returns the processx::process() object cached by vos_start() to control the external Virtuoso sever process from R.

Examples

```
if(has_virtuoso())
vos_process()
```

vos_query *Run a SPARQL query*

Description

Run a SPARQL query

Usage

```
vos_query(con, query)
```

Arguments

con	a ODBC connection to Virtuoso, from vos_connect()
query	a SPARQL query statement

Details

SPARQL is a graph query language similar in syntax SQL, but allows the use of variables to walk through graph nodes.

Value

a `data.frame` containing the results of the query

References

- https://en.wikipedia.org/wiki/SPARQL
- https://docs.ropensci.org/rdflib/articles/rdf_intro.html

See Also

`vos_start()`, `vos_connect()`

Examples

```
vos_status()

if(has_virtuoso()){
vos_start()
con <- vos_connect()

# show first 4 triples in the database
DBI::dbGetQuery(con, "SPARQL SELECT * WHERE { ?s ?p ?o } LIMIT 4")
}
```

---

vos_set_paths *set Virtuoso paths*

Description

Set the location of Virtuoso database, configure files, cache, and logs to your preferred location. Set home to the location of your Virtuoso installation.

Usage

```
vos_set_paths(
  db_dir = vos_db(),
  config_dir = vos_config(),
  cache_dir = vos_cache(),
  log_dir = vos_logdir(),
  home = virtuoso_home()
)
```

Arguments

db_dir	Location of data in the Virtuoso (tables, triplestore)
config_dir	Location of configuration files for Virtuoso
cache_dir	Location of cache for bulk importing
log_dir	Location of Virutoso Server logs
home	Location of the Virtuoso installation

Value

A logical vector, with elements being true if setting the corresponding variable succeeded (invisibly).

Examples

```
if(has_virtuoso())
  vos_set_paths()
```

---

vos_start *Start a Virtuoso Server*

Description

This function will attempt to start a virtuoso server instance that can be managed completely from R. This allows the user to easily start, stop, and access server logs and functions from the R command line. This server will be automatically shut down when R exits or restarts, or can be explicitly controlled using vos_kill(), vos_log(), and vos_status().

Usage

```
vos_start(ini = NULL, wait = 30)
```

Arguments

ini	path to a virtuoso.ini configuration file. If not provided, function will attempt to determine the location of the default configuration file.
wait	number of seconds to wait for server to come online

Details

It can take some time for the server to come up before it is ready to accept queries. vos_start() will return as soon as the server is active, which typically takes about 10 seconds on tested systems. vos_start() monitors the Virtuoso logs every one second for a maximum time of wait seconds (default 30 seconds) to see if the server is ready. If wait time is exceeded, vos_start() will simply return the current server status. This does not mean that starting has failed, it may simply

need longer before the server is active. Use vos_status() to continue to monitor the server status manually.

If no virtuoso.ini configuration file is provided, vos_start() will automatically attempt to configure one. For more control over this, use vos_configure(), see examples.

Value

invisibly returns the processx::process() object which can be used to control the external process from R. It is not necessary for a user to store this return object, as vos_start() caches the process object so it can be automatically accessed by other functions without needing to store and pass the return object.

See Also

vos_install()

Examples

```
if(has_virtuoso()){
vos_start()
## or with custom config:
vos_start(vos_configure(gigs_ram = 3))

}
```

---

vos_status *Query the server status*

Description

Query the server status

Usage

```
vos_status(p = NA, wait = 10)
```

Arguments

p	a process object, returned by vos_process() or vos_start(). (will be restored from cache if not provided)
wait	number of seconds to wait for server to come online

Details

Note: Use vos_log() to see the full log

Value

a character string indicating the state of the server:

- "not detected" if no process can be found
- "dead" process exists but reports that server is not alive. Server may fail to come online due to errors in configuration file. see `vos_configure()`
- "running" Server is up and accepting queries.
- "sleeping" Server is up and accepting queries.

Examples

```
if(has_virtuoso())
  vos_status()
```

`vos_uninstall` *Uninstall Virtuoso*

Description

Automatic uninstaller for Mac OSX and Windows clients.

Usage

```
vos_uninstall()
```

Examples

```
## Not run:
vos_uninstall()

## End(Not run)
```

Index