Package 'SQRL'

February 14, 2019

Type Package

Title Database Query Interfaces

Version 0.6.2

Date 2019-02-14

Author Mike Lee

Maintainer Mike Lee <random.deviate@gmail.com>

Description Facilitates interaction with 'ODBC' data sources.

License GPL-3

Depends R (>= 3.3.0)

Imports base, RODBC

Suggests tools, utils

NeedsCompilation no

Repository CRAN

Date/Publication 2019-02-14 05:20:03 UTC

R topics documented:

SQRL-package *Database Query Interfaces*

Description

Facilitates exploratory work and rapid prototyping with Open Database Connectivity (ODBC) data sources.

Details

Automatically creates like-named interface functions to ODBC data source names (DSNs). These functions support multi-statement SQL scripts, with or without arguments, embedded R expressions, or flow-control structures.

Additional interfaces can be defined at any time. Set-and-forget communication parameters are managed behind the scenes. The package is a wrapper about **RODBC**.

Version

0.6.2

Author(s)

Mike Lee

See Also

sqrlSources, sqrlUsage

Examples

```
require(SQRL)

# Show (automatic) data sources.
sqrlSources()

## Not run:
# If ratatoskr were one of those sources
# (i.e., if a DSN of that name was found),
# then a query could be submitted like so:
ratatoskr("select messages from vedfolnir ",
          "where addressee = 'nidhogg' ",
          "limit ", 5)

## End(Not run)

# Define a new data source.
sqrlSource("mysource",
            driver = "MYSQL ODBC 5.3 ANSI Driver",
            server = "localhost",
```

```
            user = "<uid>",
            password = "<pwd>")

## Not run:
# Obtain help on usage.
mysource("help")

# Submit a query to the new source.
mysource("select * from database.table")

# Submit a parameterised query from file.
mysource("transactions.sqrl", customerid = 111111)

## End(Not run)
```

`sqrlAll` *Broadcast a Command to All Data Sources*

Description

Passes a single command to every **SQRL** data source in turn.

Usage

```
sqrlAll(...)
```

Arguments

`...`	The command to broadcast (as per `sqrlUsage`).

Value

Returns a named list containing the result of the command for each data source. The list is invisible, except when retrieving (getting) a named parameter value.

Note

The command is passed to all **SQRL** data sources, whether or not they have defined interfaces.

The command can be a SQL query.

See Also

`sqrlOff`, `sqrlUsage`

Examples

```
# Show all interfaces (visible return).
sqrlAll("interface")

# Do not convert strings to factors.
sqrlAll("stringsAsFactors", FALSE)

# Enable all connection indicators.
sqrlAll(visible = TRUE)

# Close all open channels.
sqrlAll("close")

# Remove all defined sources.
sqrlAll("remove")
```

---

sqrlConfig *Configuration Files*

Description

This material does not describe a function, but (rather) the file format used to configure **SQRL** interfaces and **RODBC** communications.

Configuration files can be used to define new data sources, set blanket parameter values for existing sources, or set individually named parameter values.

Example Configuration File

```
# Parameters for RODBC:odbcConnect/RODBC:odbcDriverConnect.
dsn                 =
uid                 = stanislaus
pwd                 = D:/some/other/file.txt
connection          = "driver=<driver>;dbalias=quag;uid=<uid>;pwd=<pwd>;"
case                = "nochange"
believeNRows        = TRUE
colQuote            = c("`", "'")
tabQuote            = "
interpretDot        = TRUE
DBMSencoding        = ""
rows_at_time        = 100
readOnlyOptimize    = FALSE

# Additional parameters for RODBC:sqlQuery.
errors              = TRUE
as.is               = TRUE
max                 = 0
buffsize            = 1000
```

```
nullstring          = NA_character_
na.strings          = c("NA", "-", "")
dec                 = .
stringsAsFactors    = FALSE

# Parameters for SQRL.
autoclose           = TRUE
driver              = "{IBM DB2 ODBC DRIVER}"
interface           = "Q"
ping                = "select 1 from dual"
verbose             = FALSE
visible             = TRUE
prompt              = "Q"
wintitle            = "(Quag)"
```

Commentary on Example File

This is a sample configuration file, exhibiting all parameters. In general, a file need not include all of these (default values are in place).

Parameters may be defined as the path to some other file. The *driver* and *dsn* parameters will take paths as their values. For all other parameters, a value will be read from within the file. Such files may contain only a single line with only the one parameter value on it, or they may adhere to the full (multiple) 'parameter = value' format. Unexpected results may occur should a file called (say) "TRUE" exist.

File paths should not be wrapped in quotes. Other strings may be wrapped, although this is not mandatory unless they clash with the name of an object within the R base namespace.

When the right hand side of the equals is left blank, no action is taken (the parameter on the left hand side is not assigned to).

See Also

sqrlParams, sqrlSource

Examples

```
# Define a new source (not from file).
sqrlSource("Orac", "Dbq=Delphi;Uid=Pythia;Pwd=<pwd>",
           "Driver={Oracle in OraClient11g_home1}")

# Review the current configuration (parameter values).
Orac("config")

## Not run:
# Configure an existing source from file.
Orac("config", "my/config/file")

# Set one named parameter from file.
Orac(connection = "my/other/file")
```

```
# Define and configure a new source from file.
sqrlSource("Zen", "my/other/", "file")

## End(Not run)
```

`sqrlInterface` *Creates Data Source Interfaces*

Description

Creates, renames, and/or removes data source interface functions.

Communications with data sources (including SQL queries) are conducted through these interfaces.

Usage

```
sqrlInterface(...)
```

Arguments

...	The name of a defined data source, and the name to use for its interface.

Details

The source and interface names may be supplied as two character strings, ("source", "interface"), or as a (source = "interface"), or ("source" = "interface"), pair.

The setting of an interface whose name would clash with that of any other object already on the R search path is prevented. An error will be thrown if a potential conflict is detected. Conversely, a successful call of this function guarantees both the existence of the new interface, and the uniqueness of its name.

If the interface name is specified as either NULL or "remove", then any existing interface is deleted (and no new interface is created).

If only a single string is supplied, the name of that source's interface function is returned.

Value

Returns the name of the source's interface function (visibly on get, invisibly on set).

Note

Interfaces are stored in a publicly accessible environment, SQRL:Face. This is attached to the R search path when the package is loaded.

See Also

sqrlSource

Examples

```
# Define a new data source, named 'entropy'.
sqrlSource("entropy", uid = "ludwig", pwd = "<pwd>",
           driver = "{SQL Server Native Client 11.0}",
           server = "Clausius", database = "Gibbs")

# The source comes with an interface of the same name.
sqrlInterface("entropy")
entropy("sources")

# Change the name of the interface function.
sqrlInterface("entropy", "S")

# The name of the source remains unchanged.
sqrlInterface("entropy")
S("sources")

## Not run:
# Submit a query, via the interface.
S("select 1")

# Submit a script, via the interface.
S("My/SQL/file.sql")

# Submit a parameterised script, via the interface.
S("My/SQRL/file.sqrl", month = "April")

## End(Not run)

# Remove the source's interface function.
sqrlInterface("entropy", "remove")

# The source remains, but has no interface.
sqrlInterface("entropy")
sqrlSources()
```

---

sqrlOff *Close Connections and Deactivate Package*

Description

Closes all connections, detaches the interface environment (*SQRL:Face*) from the search path, and unloads the **SQRL** namespace. No further communication with any data source will be possible through **SQRL** (unless it is reloaded).

Usage

```
sqrlOff()
```

Value

Returns invisible NULL.

Note

Calls to RODBC:odbcCloseAll will close any connection channels open in **SQRL**.

See Also

SQRL

Examples

```
## Not run:
# Calling sqrlOff() will deactivate and unload SQRL.
sqrlOff()

## End(Not run)
```

---

sqrlParams *Control and Connection Parameters*

Description

This material does not describe a function, but (rather) the various parameters governing ODBC communications and package behaviour. The majority of these are passed through to **RODBC**.

Parameters

as.is: A logical vector, or a numeric vector (of column indices), or a character vector (of column names). Argument to RODBC:sqlQuery (see also utils:read.table). Tells **RODBC** which character columns of a table, as returned by a query to the ODBC connection, *not* to convert to some other data type (i.e., which character columns to leave as is). Due to **SQRL**'s set-and-forget approach to parameters, it is inconvenient to change *as.is* on a query-by-query basis. That being the case, it is usually defined as a logical singleton (either TRUE or FALSE). Default value is FALSE (convert all character columns).

autoclose: A logical singleton. Tells **SQRL** whether or not to automatically close the data source connection after each query (in general, a sequence of multiple statements). The default value is FALSE, which leaves the connection open. When set to TRUE, connections will normally only open for the duration of each query, but may sometimes remain open if a mid-query exception is thrown. Setting TRUE has no immediate effect upon a connection that is already open. When user input is required for authentication each time a new connection is opened, the default setting will be more convenient.

believeNRows: A logical singleton. Argument to RODBC:odbcDriverConnect. Tells **RODBC** whether or not to trust the nominal number of rows returned by the ODBC connection. Locked while the connection is open. Default value is TRUE.

buffsize: A positive integer. Argument to RODBC:sqlQuery. Specifies the number of rows (of a query result) to fetch at a time. Default value is 1000.

case: A character string, specifically one of "nochange", "toupper", "tolower", "mysql", "postgresql", or "msaccess". Argument to RODBC:odbcDriverConnect. Specifies case-changing behaviour for table and column names. Locked while the connection is open. Default value is "nochange".

channel: An **RODBC** connection handle. Returned by RODBC:odbcDriverConnect. Argument to RODBC:sqlQuery. This parameter is read-only.

colQuote: A character vector of length 0, 1, or 2, or NULL. Argument to RODBC:odbcDriverConnect. Specifies the quote character(s) for column names. A vector of length zero means no quotes, of length one means apply the specified quote character at both ends of a name, and of length two means apply the first specified character to the start of the name and the second specified character to the end of the name. Locked while the connection is open. The default value is a backtick for MySQL, and a double-quote for everything else.

connection: A character string. Argument to RODBC:odbcDriverConnect. Specifies an ODBC connection string. The content of this string will be database-management system (DBMS) dependent. Overrides *dsn*, should both be defined. Locked while the connection is open. Defaults to the empty string (connect via DSN instead). Will accept NULL as an alias for the empty string. Setting *connection* resets *dsn*, unless *connection* contains the "<dsn>" placeholder.

DBMSencoding: A character string. Argument to RODBC:odbcDriverConnect. Names the encoding returned by the DBMS. Locked while the connection is open. Default value is the empty string (use encoding of the R locale). Will accept NULL as an alias for the empty string.

dec: A character string (typically a single character). Argument to RODBC:sqlQuery. Defines the decimal-place marker to be used when converting data from text to numeric format. The default value is options("dec"), as set by **RODBC**.

driver: A character string. The name or file path of the ODBC driver for the source (either currently in use, or to be used when a channel is opened). This determines the requisite dialect of SQL. Locked while the connection channel is open. Defaults to the empty string. Will accept NULL as an alias for the empty string.

dsn: A character string. Argument to RODBC:odbcConnect. Specifies the data source name (DSN) to connect to. Can be a file path. Overridden by *connection*, when that parameter is defined. Setting *dsn* resets *connection*, unless *connection* contains the "<dsn>" placeholder. Setting *dsn* also sets *driver*, if the DSN exists and the associated driver can be identified. Locked while the connection is open. Defaults to the empty string. Will accept NULL as an alias for the empty string.

errors: A logical singleton. Argument to RODBC:sqlQuery. Controls whether or not to throw R errors in response to DBMS/ODBC exceptions. Default value is TRUE (this differs from the **RODBC** default).

interface: A character string, or NULL. The name of the **SQRL** interface function for this data source (see sqrlInterface). Setting NULL or "remove" removes the interface. Default value is NULL (undefined).

interpretDot: A logical singleton. Argument to RODBC:odbcDriverConnect. Locked while the connection is open. Controls whether or not to interpret table names of the form "aaa.bbb" as table "bbb" in schema/database "aaa". The default value is TRUE.

max: An integer. Argument to RODBC:sqlQuery. Caps the number of rows fetched back to R. The default value is 0 (meaning unlimited; retrieve all rows).

na.strings: A character vector. Argument to RODBC:sqlQuery. Specifies strings to be mapped to NA within character data. The default value is "NA".

name: A character string. The name of this **SQRL** data source. While often identical to the names of both the underlying ODBC data source and the **SQRL** interface function, it need match neither in general. Multiple **SQRL** sources may interface with the same ODBC source. This parameter is write once, and cannot be changed after creation of the **SQRL** source. There is no default value.

nullstring: A character string. Argument to RODBC:sqlQuery. The string with which to replace SQL_NULL_DATA items within character columns. The default value is NA_character_.

ping: A character string. Defines a simple SQL statement used by **SQRL** to verify source connections. This enables **SQRL** to make one automatic reconnection attempt after a network outage or other unexpected channel closure (this process is silent unless user-input is required for authentication). The default value is DBMS-dependent. Manual definition may be necessary in the event that **SQRL** fails to identify an appropriate statement for the particular DBMS of the source.

prompt: A character string (typically a single character). Defines an indicator to be applied to the R command prompt when the connection is open and *visible* is TRUE. Defaults to the first character of *name*. Single-letter indicators are recommended since, if two sources are open and one indicator is a substring of the other, then **SQRL** may fail to correctly update the prompt when one source is closed. Can be set to an empty string, in which case nothing is applied to the prompt. Will accept NULL as an alias for the empty string.

pwd: A character string. Argument to RODBC:odbcConnect. Specifies the password for authentication. Locked while the connection is open. Defaults to the empty string (interpreted as do not supply a password to the ODBC driver). Will accept NULL as an alias for the empty string. Write-only.

readOnlyOptimize: A logical singleton. Argument to RODBC:odbcDriverConnect. Specifies whether or not to optimise the ODBC connection for read-only access. Locked while the connection is open. Default value is FALSE.

rows_at_time: A positive integer, between 1 and 1024. Argument to RODBC:odbcDriverConnect. Specifies the number of rows to fetch at a time when retrieving query results. Locked while the connection is open. The default value is 100. Manually setting 1 may be necessary with some ODBC drivers.

stringsAsFactors: A logical singleton. Argument to RODBC:sqlQuery. Controls the conversion of character to factor columns within query results, excluding those columns covered by *as.is*. Defaults to default.stringsAsFactors().

tabQuote: A character vector of length 0, 1, or 2, or NULL. Argument to RODBC:odbcDriverConnect. Specifies the quote character(s) for table names. A vector of length zero means no quotes, of length one means apply the specified quote character at both ends of a name, and of length two means apply the first specified character to the start of the name and the second specified character to the end of the name. Locked while the connection is open. Defaults to the value of *colQuote*.

uid: A character string. Argument to RODBC:odbcConnect. Specifies the user identity (UID, user name) to use on the data source. Locked while the connection is open. Defaults to the local name of the R user (Sys.info()["user"]). Will accept NULL as an alias for the empty string (which is interpreted as do not pass a UID to the ODBC driver). May be inaccurate when the UID is specified within a DSN.

verbose: A logical singleton. Controls whether or not to display verbose output while processing SQL or SQRL files. The default is FALSE (verbose output disabled). When TRUE, usually only the head of each intermediate object is displayed. Sufficiently exotic objects may fail to print.

visible: A logical singleton. Toggles display of the *wintitle* and *prompt* indicators (while an open connection channel exists to the source). The default value is FALSE (do not show indicators). Changing this to TRUE authorises modification of the "prompt" global option (see base:options).

wintitle: A character string, possibly empty. Will accept NULL as an alias for the empty string. Defines an indicator that, unless empty, is displayed on the R window title bar while a connection channel is open to the source, and provided *visible* is TRUE. An asterisk (*) is appended to the indicator while a query is running on the source (including connection-testing pings), and a plus-sign (+) is appended while results are being retrieved from it. If two sources are open and one indicator is a substring of the other, then **SQRL** may fail to correctly update the title when one source is closed. Only works with 'R.exe', 'Rterm.exe' and 'Rgui.exe', and then only while running on a "Windows" operating system. Works with both MDI and SDI modes, but does not work with "RStudio".

Note

Each **SQRL** data source has its own set of the above parameters. Altering the value of a parameter (e.g., *stringsAsFactors*) for one source does not affect the value for any other source. Use sqrlAll() to make blanket changes.

See Also

sqrlUsage, **RODBC**

---

sqrlScript *Combined Language Scripts*

Description

This material does not describe a function, but (rather) the SQRL script file format for SQL with embedded R.

For instructions on how to submit (run) these scripts from file, refer to sqrlUsage.

The following (very simple) example scripts won't necessarily work with your own version of SQL or your own data source.

Example Script #1 (Multiple Statements)

```
-- My file
select 1;
select 2;
select 3;
```

Commentary on Example Script #1: Multi-statement SQL files can be copied directly from "SQL Developer" (or similar application). Each of the (above) three queries will be submitted in turn. Only the final result will be returned to the R calling environment (in this case, a data frame containing a single value of 3).

Example Script #2 (Embedded R)

```
-- This is SQL
select
  <R>
    # This is R
    if (exists("x")) {
      x
    } else {
      0
    }
  </R>
/* This is SQL */
from
  dual
```

Commentary on Example Script #2: **SQRL** supports the embedding of R within SQL, via XML-style tags (<R> ... </R>), as above. This enables (explicitly and/or implicitly) parameterised SQL queries.

In this example (above), if the variable *x* was explicitly passed to the query, then the supplied value will be used (see sqrlUsage). If *x* was not explicitly passed, then it can be inherited (implicitly passed) from the calling environment. In this case, if *x* was not explicitly supplied and also does not *x* exist within the calling environment, then a default value of 0 will be used.

SQL comments are allowed within R (<R> ... </R>) sections, so that SQL syntax highlighting can be better applied to the file. R comments are not allowed within SQL sections

Nested SQL queries can be made from within <R> ... </R> tags. However, <R> ... </R> tags cannot be nested.

Example Script #3 (Manipulation and Feedback)

```
-- SQL statement, ending on a semicolon.
use mydatabase;

/* SQL statement, ending on a <do> tag. */
create table mynewtable
as select columnA, columnB
from originaltable
<do>

-- SQL query, ending on a <result> (assignment) tag.
-- In this case, the result of the query (a data frame)
-- is assigned to an R variable 'a' (within a temporary
-- working environment communal to this script).
select max(columnA)
```

```
from mynewtable
<result -> a>

# Manipulate the result in R, then
# proceed to the next SQL statement.
b <- runif(1)
a <- as.numeric(a) + b
<do>

-- Use the earlier result in a new query.
select columnA, columnB from mynewtable
where columnA > <R> a * b </R>
and columnB < <R> z <- 5; z + 2 </R>
<result -> x>

-- Only the result of the final operation
-- is returned at the end of the script.
return(list(minA = min(x$columnA),
            maxB = max(x$columnB))
```

Commentary on Example Script #3: SQL statements are terminated by any one of a semicolon, a <do> tag, a <result> tag, or the end of the file. After a <result> tag, statements are interpreted as R (as opposed to SQL) until either a <do> tag, or the end of the file, is encountered. Following a <do> tag, statements are interpreted as SQL.

In the case of a <result -> var> tag, the SQL query result is assigned to an R variable *var* within the script's temporary working environment. This variable can be arbitrarily manipulated in R, and the result of that manipulation used as part of a subsequent SQL query.

With the exception of the name of the target (assignment) variable within a result tag, tags are not case sensitive. When the name of the target variable is either "null" or "NULL", then the query result is not assigned into the working environment (as with a <do> tag), but the subsequent script is still interpreted as R (unlike with a <do> tag).

After the end of the script, the working environment is lost, along with any variables within it. Only the final result is returned to the calling environment.

Example Script #4 (R and Lists)

```
-- R sections can be used to set temporary variables.
<R>
  columns <- list("columnA", "columnB")
  wordlist <- list("'red'", "'blue'")
  conditions <- c("and columnA < 2", "and columnD > 4")
<do>

-- Now the query.
select
  <R> columns </R>
from
  dbname.tablename
where
```

```
  columnC in (<R> wordlist </R>)
  <R> conditions </R>

-- End the script here.
-- The statements below are ignored.
<stop>
  and columnA > 0
```

Commentary on Example Script #4: Whereas <R> ... </R> sections are embedded within SQL (which may be blank), <R> ... <do> sections exist outside of any SQL. While the result of an <R> ... </R> section is substituted back into the surrounding SQL (and will form part of the query), an <R> ... <do> section is simply evaluated within the script's working environment. Although a <result -> null> ... <do> section may be preceded by SQL (which will be submitted prior to evaluating the section), an <R> ... <do> section may not be (the two sections being otherwise equivalent).

Lists are inserted comma collapsed. Vectors are inserted newline collapsed.

A <stop> tag imitates the end of the file, and can be used to interrupt a script for debugging.

Example Script #5 (If, While, and Return)

```
/* Implements drop-if-exists in a SQL that doesn't. */

-- Pull details of temporary tables.
help volatile table
<result -> a><do>

-- If those details are a data frame, then
-- at least one temporary table exists.
<if (class(a) == class(data.frame()))>
  <R> i <- 1 <do>

  -- Drop each table in turn.
  <while (i <= nrow(a))>
    drop table <R> a[i, "Table Name"] </R> <do>
    <R>
      print(paste("dropped", a[i, "Table Name"]))
      i <- i + 1
    <do>
  </while>

  -- Exit from here.
  <return (paste(i - 1, "tables dropped"))>
</if>

-- Otherwise, we had no temps to drop.
<return ("no temps found")>
```

Commentary on Example Script #5: Within SQL sections, <if> and <while> tags can be used to control the (repeated) submission of queries. The parentheses enclosing the conditional R

expression are mandatory, and that expression must evaluate to a Boolean singleton.

Within SQL sections, <return> tags can be used to exit early from a script. A return value (R expression) is mandatory, and it must be enclosed in parentheses. Calling the base return function inside an R section does not exit the script; it merely sends a value to the temporary working environment.

A difference between <return> and <stop>, is that the latter ignores conditionals; <stop> applies even when nested within an <if> that evaluated to FALSE.

Example Script #6 (Compact Variant)

```
help volatile table <result -> a>;
<if (i <- 0; class(a) == class(data.frame()))>
  <while (i <- i + 1; i <= nrow(a))>
    drop table <R> a[i, "Table Name"] </R>;
    <R> print(paste("dropped", a[i, "Table Name"]));;
  </while>
  <return (paste(i - 1, "tables dropped"))>
</if>
<R> "no temps found"
```

Commentary on Example Script #6: This is an alternative implementation of example #5.

Multiple R statements are allowed within conditional tag expressions. In the case of while loops, all statements within the tag expression are evaluated on each iteration. The condition takes the value of the final statement, which must evaluate to either TRUE or FALSE.

Semicolons can be used to terminate R sections (as well as SQL statements). Within an R section (following either an <R> or a <result> tag), if there is nothing but whitespace (or nothing at all) between a semicolon and the start of its line, the previous semicolon, or the start of the R section (whichever comes first, heading left from the semicolon), then that semicolon marks the end of the R section and is equivalent to a <do> tag (the subsequent script is read as SQL). So located semicolons are not accepted by R's parser.

Example Script #7 (Else and Else If)

```
select
  <if (dow == "Monday")>
    ColumnA
  <else if (dow %in% c("Saturday", "Sunday"))>
    ColumnB, ColumnC
  <else>
    *
  </if>
from
  some.table
```

Commentary on Example Script #7: The closing tags, </if> and </while>, do not imply <do>. This allows switching column names, or other SQL fragments, in and out of a larger query.

The same end can be achieved through embedded R (<R> ... </R> tags), or other mechanisms. In the above, *dow* is an externally supplied parameter (see example #2).

The parser is simple, and does not enforce correct nesting structure. Unintuitive output may appear when nesting violations occur.

Note

The *verbose* parameter toggles extended (intermediate) output when running scripts (see sqrlParams).

See Also

sqrlUsage

Examples

```
# Define a new data source.
sqrlSource("mire", "dsn=Mire")

## Not run:
# Submit a SQL script to the source.
mire("my/script.sql")

# Submit a SQRL script, with explicit parameter values.
mire("my/parameterised/script.sqrl",
      day = Sys.Date(), colour = "'blue'")

## End(Not run)
```

---

sqrlSource *Define New Data Sources*

Description

Defines new data sources and creates the interface functions for communicating with them. Can also redefine and/or delete existing sources.

Usage

```
sqrlSource(...)
```

Arguments

...	A name and definition for the source (refer to the details section, below).

Details

The arguments may be supplied as a single *name = definition* pair, or as a sequence of unnamed strings. In the latter usage, the leading string is taken for the new source *name*, while the remainder constitute its *definition*. In either usage, the *definition* may be a connection string, a configuration file, a data source name (DSN), or the name of an existing **SQRL** source. To be precise, if the

definition specifies the path of an existing file, then the source is configured from that file. Otherwise, if the *definition* names an existing source, then the new source is created as a copy of that. Otherwise, if the *definition* consists of multiple terms, or if it contains an equals character (=), then it is assumed to represent a connection string. When none of these conditions apply, the *definition* is assumed to specify a DSN.

Alternatively, the arguments may instead be supplied as a single string followed by at least one named *parameter = value* pair. In this usage, the leading string is taken for the new source name, while the named pairs constitute its definition. The named pairs may represent components of a connection string, and/or **SQRL** parameter values to be applied (see sqrlParams). If one of the pairs is named "copy", then the new source is created as a copy of the corresponding preexisting source. If one of the pairs is named "config", then the new source is configured from the corresponding file (see sqrlConfig). If neither of those applies, and if no pair is named "connection" (i.e., if no explicit value was given for the connection string), and if either the remaining names do not include "dsn", or they do include some other name not recognised as being a **SQRL** parameter, then those pairs are concatenated into a connection string.

Whichever form is used, the new interface name (which defaults to the source name) must not conflict with that of any other object on the R search path (or else an error will be thrown).

Redefinition of an existing source is allowed, provided it is closed.

When the source name is 'remove', the definition is interpreted as a list of sources to be deregistered. This precludes the use of 'remove' as a source name. Alternatively, redefining a source to NULL also deregisters the source.

Value

Invisibly returns the interface function name, after defining the new source and creating that interface. In general, this need not match the name of the source itself.

Note

Source definitions are not checked for validity (specified connection strings need not be correct, specified DSNs need not exist).

Connection strings may include placeholders; "<dsn>", "<driver>", "<uid>", and "<pwd>", to be replaced with the corresponding parameter values on the opening of a channel. These placeholders are case sensitive (see sqrlParams).

In 'Rgui.exe', the ODBC driver may, via **RODBC**, prompt for missing connection details (username, password, etc.). In other R applications, those details will need to be complete (no prompting occurs).

See Also

sqrlConfig

Examples

```
# Define a new source from a DSN.
sqrlSource("daedalus", dsn = "Knossos")

# Define another source as a copy of the former.
```

```
sqrlSource(icarus = "daedalus")

# Redefine an existing source by a connection string.
# (This example is for a Windows-system client.)
sqrlSource("icarus",
            driver = "PostgreSQL ANSI(x64)",
            server = "localhost",
            port = 5432,
            uid = "asterion",
            pwd = "moo")

# Define a new source by a connection string.
# (This example is for a GNU/Linux-system client,
#  and employs the <pwd> password placeholder.)
sqrlSource("knossos",
            "dbcname=Knossos;uid=theseus;pwd=<pwd>",
            "driver=/opt/teradata/client/16.10/lib64/tdata.so")

## Not run:
# Define a new source from a configuration file.
sqrlSource(minos = "path/to/minos.config")

# Define a new source, ariadne, as a copy of the existing
# source, minos, then apply the configuration file conf.txt
# over that, and then set both the connection string and
# interface function name (parameter values) over those.
sqrlSource("ariadne",
            copy = "minos",
            config = "conf.txt",
            connection = "DSN=Knossos",
            interface = "a")

# Source names may conflict with those of preexisting
# R objects, provided that the config file defines a
# conflict-free name for the source's interface function.
sqrlSource("c", "path/to/", "c.config")

## End(Not run)

# Review defined sources.
sqrlSources()

# Remove two of the sources.
sqrlSource("remove", c("daedalus", "knossos"))

# Remove another.
sqrlSource(icarus = NULL)
```

---

sqrlSources *List Data Sources and their Interfaces*

Description

Returns a summary of defined data sources. These will consist of system and user DSNs, plus any additional sources defined via sqrlSource.

Usage

```
sqrlSources(...)
```

Arguments

...	An optional character string. If set to one of "all", "user", or "system", then a call is made to RODBC:odbcDataSources (with the corresponding *type* value) to re-examine that class of data source names (DSNs) and import all those found. If set to "remove", then all currently defined sources are deregistered.

Value

Returns a data frame of data source details.

Note

The return frame may have zero rows, if no data sources are defined.

Sources need only to have been defined; they need not actually exist.

DSNs with "Access", "dBASE", or "Excel" in their names are not automatically imported. They can be manually added via sqrlSource.

See Also

sqrlInterface, sqrlSource

Examples

```
# Review defined sources.
sqrlSources()

## Not run:
# Sample sqrlSources() output:

   name interface open                    driver
1 chaos     chaos    N       PostgreSQL ANSI(x64)
2 order      <NA>    N MySQL ODBC 5.3 ANSI Driver

# Here, there are two data sources; 'order' and 'chaos'.
# The interface to 'chaos' is a function of the same name.
# No interface has yet been defined for 'order' (use of
# that name is prevented due to its conflicting with the
# base:order function). Neither source (channel) is open.

## End(Not run)
```

```
# Remove all sources.
sqrlSources("remove")

# Reload user DSNs.
sqrlSources("user")
```

---

sqrlUsage *How to Use the Interface Functions*

Description

This material does not describe a single function, but (rather) how to use **SQRL** interfaces, once created. These functions do not have their own help files, since their names are not pre-determined.

Details

Once you have a named interface, created either automatically (on loading of the **SQRL** namespace) or manually (via `sqrlSource()`), it can be used to communicate with the associated data source. Connection handles and communication parameters are managed under the hood.

Subsequent sections provide usage examples for an interface called `thoth`. The names of your own interface functions can be discovered by calling `sqrlSources()`.

Opening and Closing

```
# Open a connection to the data source.
thoth()

# Alternative method (explicit form).
thoth("open")

# Doing this is fine (the channel survives).
rm(list = ls(all.names = TRUE))

# Check if the connection is open.
thoth("isopen")

# Open a connection and confirm status.
thoth()$isopen

# Close the connection.
thoth("close")

# Close the connection when not in use.
thoth(autoclose = TRUE)
```

Opening connections in the above way isn't usually necessary, since this occurs automatically as required.

The isopen command "pings" the data source, to reliably establish whether or not the connection really is open (including after a network outage or remote closure).

With autoclose = TRUE, isopen will normally return FALSE, since the connection is closed after every data source command sequence (including the open command).

Submitting Queries

```
# Submit a query.
thoth("select 1")

# Submit another query.
thoth("select ", sample(6, 1), " from dual")

# Submit a query from file.
thoth("my/", "file.sql")

# Submit a parameterised query from file.
thoth("emissions.sqrl", make = "VEB", model = "Trabant 601")

# Submit a multi-statement query.
thoth(query = "use necronomicon; select top
               <R> N </R> shoggoths from pit", N = 5)

# Ensure input is treated only as a file name.
thoth(file = "create table")
```

If necessary, a connection channel will be opened beforehand. The connection will remain open afterwards, unless *autoclose* is TRUE.

To be clear, the phrase 'parameterised query' is not meant in the sense of prepared or parameterised statements (as per package **RODBCext**). Here, parameter substitution occurs inside R, with the resulting string being passed to the ODBC driver as an ordinary query.

If a query should fail because of an unexpectedly lost connection, an attempt will be made to re-connect and re-submit. Unless credentials are required for authentication, this should go unnoticed by the user.

When a query returns no data (as would 'use database'), the interface function returns invisibly.

If a file called (say) 'use database' should exist, then thoth("use database") submits the content of that file (rather than the apparent command). This renders use of the *file* argument mostly unnecessary.

Use of either the *query* or *file* argument forces interpretation of the corresponding value as a query or file path (from which to read a query), respectively. The *query* argument uses the *verbose* option and allows multiple statements and embedded R (as with sqrlScript files), whereas unnamed queries do not (instead, they allow and concatenate multiple strings).

Querying Metadata

```
# Get information on source data types.
thoth("typeinfo")
```

```
# List all tables.
thoth("tables")

# List all tables within a database (schema).
thoth("tables", "mydatabase")

# Get information on the columns of a particular table.
thoth("columns", "my.table")
```

The `typeinfo`, `tables`, and `columns` commands are simple (reduced functionality) wrappers about `RODBC:sqlTypeInfo`, `RODBC:sqlTables`, and `RODBC:sqlColumns` (respectively).

Reviewing Settings

```
# Get the associated source definition.
thoth("source")

# Get the value of one named parameter.
thoth("uid")

# Alternative method (pings the source).
thoth()$uid

# List the values of all parameters.
thoth("config")
```

Passwords are returned obliterated.

Setting Parameters

```
# Enable visible indication of open connections.
thoth("visible", TRUE)

# Define the ping statement for the data source.
thoth("ping", "use database")

# Do not convert strings to factors.
thoth("stringsAsFactors FALSE")

# Set (opening and closing) table-name quotes.
thoth(tabQuote = c("`", "'"))

# Setting multiple parameters at once.
thoth(as.is = TRUE, na.strings = c("NA", "-", ""))

# Set one (named) parameter from a file.
thoth("ping" = "path/to/file")

# Import an entire configuration file.
thoth("config", "path/", "to/", "file")
```

```
# Reset parameters to their default values.
thoth("reset", c("as.is", "na.strings"))
```

Calls of the form `thoth("name value")`, `thoth("name", "value")`, `thoth("name", value)`, `thoth(name = value)`, `thoth("name" = value)`, `thoth(name = "value")` and `thoth("name" = "value")` are largely interchangeable.

The *driver* and *dsn* parameters accept file paths as their values. For all other parameters, values are extracted from within any specified files.

Assigning *visible* `TRUE` authorises modification of the global `prompt` option. When running '`R.exe`', '`Rterm.exe`' or '`Rgui.exe`' on a "Windows" operating system, this also authorises modification of the R window title.

Changing the Interface

```
# Change the interface.
thoth("interface", "H")

# Change it back.
H(interface = "thoth")
```

If the proposed new interface name already belongs to some other object within the R search path, then the change request will be denied (unless that name is "remove", in which case the current interface function will be deleted).

A successful change deletes the previous interface.

Listing Data Sources

```
# See the data sources and their interfaces.
thoth("sources")
```

This is equivalent to calling `sqrlSources()`.

Getting Help

```
# Get help on 'thoth'.
thoth("help")

# Alternative form.
thoth("?")
```

The above calls will attempt to provide help tailored for the specific interface, and will fall back to these notes (`help(sqrlUsage)` or `?sqrlUsage`) should that fail.

Either of the commands `text` or `html` may be appended to `help` if a specific output format is required.

Removing the Source

```
# Deregister the associated source.
thoth("remove")
```

This closes any open connection to the data source, deletes the (thoth) interface function, and deregisters the source from **SQRL**.

See Also

sqrlAll, sqrlConfig, sqrlParams, sqrlScript

Index