Package: ODB (via r-universe)

October 16, 2024

Type Package

Title Open Document Databases (.odb) Management

Version 1.2.1

Date 2020-03-10

Author Sylvain Mareschal

Maintainer Sylvain Mareschal <maressyl@gmail.com>

URL http://bioinformatics.ovsa.fr/ODB

BugReports https://github.com/maressyl/R.ODB/issues

Description Functions to create, connect, update and query 'HSQL' databases embedded in Open Document Databases files, as 'OpenOffice' and 'LibreOffice' do.

License GPL (>= 3)

Depends methods, DBI, RJDBC

Imports utils

SystemRequirements zip

NeedsCompilation no

Repository CRAN

Date/Publication 2020-03-11 12:40:08 UTC

Contents

ODB-package *Open Document Databases (.odb) Management*

Description

Functions to create, connect, update and query 'HSQL' databases embedded in Open Document Databases files, as 'OpenOffice' and 'LibreOffice' do.

Details

The DESCRIPTION file:

Package:	ODB
Type:	Package
Title:	Open Document Databases (.odb) Management
Version:	1.2.1
Date:	2020-03-10
Author:	Sylvain Mareschal
Maintainer:	Sylvain Mareschal <maressyl@gmail.com>
URL:	http://bioinformatics.ovsa.fr/ODB
BugReports:	https://github.com/maressyl/R.ODB/issues
Description:	Functions to create, connect, update and query 'HSQL' databases embedded in Open Document Data
License:	GPL (>= 3)
Depends:	methods, DBI, RJDBC
Imports:	utils
SystemRequirements:	zip

Index of help topics:

ODB-class	Class "ODB"
ODB-package	Open Document Databases (.odb) Management
difftimeFmt	Formats a time difference in multiple units
isClosed	Checks if a DBI connection is closed
odb.close	Closes an "ODB" connection and updates the .odb file
odb.comments	Gets or sets column comments in an ODB database
odb.create	Creates a .odb empty file.
odb.export	Exports an ODB database to a SQL file.
odb.insert	Wrapper for inserting data in an ODB table.

First notice this package is not intented to manage all .odb variations. Currently two distinct types of .odb files can be distinguished : files embedding an HSQL database, and files connecting to a remote database engine such as MySQL. This package is made for the formers, as remote databases can be queried directly using the DBI package. Functions involving the XML content of the .odb file (odb.comments and odb.queries) may be operative on such files, but there is no garantee they would be.

You should not consider this package without minimal knowledge of the SQL language, as only a few basic operations have functions not relying on the user to build a valid SQL query (odb.insert, odb.tables, odb.comments, odb.queries).

This package is built around the odb class, which extends DBIConnection. A set of functions is provided to execute reading (SELECT ...) or writing (INSERT, CREATE, DROP ...) SQL queries through the HSQLDB engine, and a set of convenient tools is also provided for the most common tasks (odb.read, odb.write, odb.insert). A few Open Document specific functions are also provided, allowing modifications on features provided by the .odb file format and not the database itself (odb.comments and odb.queries).

Effects of all these functions can be achieved manually via the OpenOffice or LibreOffice Base software, obviously this package offers a way of automating it on a bigger scale. Both manual and automatic modifications can be done on the same .odb file, but users are strongly recommended to not use the two methods simultaneously. This R package works on a copy of the file made at the odb.open call, and this copy will replace the original file at the odb.close call, regardless of what happened between the two calls.

Additional tools are also provided, notably odb.export and isClosed.

See the 'examples' section for more details on the package behavior.

Author(s)

Sylvain Mareschal

Maintainer: Sylvain Mareschal <maressyl@gmail.com>

Examples

```
## CONNECTION ##

# Creation of a new .odb database
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")

# Connection to the file
odb <- odb.open(odbFile)
```

```
## STRUCTURE ##

# Creation of a new table
SQL <- "CREATE TABLE fruits (id INTEGER PRIMARY KEY, value VARCHAR(8))"
odb.write(odb, SQL)

# Addition of comments on columns
odb.comments(odb, "fruits", c("id", "value")) <- c("Fruit rank", "Fruit name")

# Printing of the database structure
print(odb.tables(odb))

# Printing of a specific comment
print(odb.comments(odb, "fruits", "value"))


## DATA ##

# Insertion of data
data <- data.frame(id=1:3, value=c("apple", "orange", "pear"))
odb.insert(odb, tableName="fruits", data=data, execute=TRUE)

# Reading
print(odb.read(odb, "SELECT * FROM fruits WHERE id=2"))

# Saved query
odb.queries(odb, "2nd record") <- "SELECT * FROM fruits WHERE id=2"
print(odb.read(odb, odb.queries(odb, "2nd record")))

# SQL export
sqlFile <- tempfile(fileext=".sql")
odb.export(odb, sqlFile)


## DISCONNECTION ##

# Is the connection closed ?
print(isClosed(odb))

# Save changes in the .odb file
odb.close(odb, write=TRUE)

# And now ?
print(isClosed(odb))
```

---

difftimeFmt *Formats a time difference in multiple units*

Description

This function converts a time difference (in seconds) into a string with various time units (days, hours, minutes, seconds and milliseconds).

Usage

```
difftimeFmt(x)
```

Arguments

x	Time difference to format (single value). Can be a floating number of seconds, or a difftime object.

Value

Returns a character of length 1, according to the following format : "[d]d [h]:[m]:[s].[ms]". Days and milliseconds are omitted if they can be.

Author(s)

Sylvain Mareschal

Examples

```
# Full format
difftimeFmt(94521.125846)

# With dates
T1 <- strptime("2011-03-21 14:32:03", format="%Y-%m-%d %H:%M:%S")
T2 <- strptime("2011-08-06 18:14:21", format="%Y-%m-%d %H:%M:%S")
dT <- as.double(difftime(T2, T1, units="secs"))
difftimeFmt(dT)
```

---

isClosed *Checks if a DBI connection is closed*

Description

Checks if a connection inheriting from the DBIConnection class (including 'ODB' connections) is closed.

Usage

```
isClosed(connection)
```

Arguments

connection	An object inheriting from DBIConnection class.

Value

Returns a single logical value, whether SQL queries can be sent to the connection or not.

Author(s)

Sylvain Mareschal

See Also

odb.open, odb.close

dbConnect, dbDisconnect

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
ODB <- odb.open(odbFile)

# Is the connection closed ?
print(isClosed(ODB))

# Save changes in the .odb file
odb.close(ODB, write=TRUE)

# And now ?
print(isClosed(ODB))
```

ODB-class *Class "ODB"*

Description

A connection to an Open Document Base (.odb) embedded HSQL database.

This class extends JDBCConnection, and objects from it can be accessed via the DBI interface. Objects from this class also store file paths used for other ODB manipulation (such as queries or comments, stored in the embedded content.xml file rather than the HSQL database).

Objects from the Class

Objects can be created by calls to the odb.open function, and saved into the original .odb file by odb.close.

Several objects can but should not refer to the same .odb file: as they deal with distinct copies of this file, updates in one connection will not propagate to others. Moreover, when one of these connections will be closed, it will override changes in the .odb file made by formerly closed connections.

Slots

`jc`: Inherited from "JDBCConnection"

`identifier.quote`: Inherited from "JDBCConnection"

`directory`: Single character value, the path to the temp directory storing the database files.

`odbFile`: Single character value, the path to the .odb file handled.

`odbVersion`: Single character value, HSQLDB version of the .odb file.

`jarFile`: Single character value, the path to the HSQLDB .jar library used.

`jarVersion`: Single character value, HSQLDB version of the .jar library.

Extends

- Class "JDBCConnection", directly.
- Class "DBIConnection", by class "JDBCConnection", distance 2.
- Class "DBIObject", by class "JDBCConnection", distance 3.

Methods

show `signature(object = "ODB")`

See also inherited methods.

Note

Functions from the `DBI` package can be used directly on the object, if you prefer to manage SQL transactions yourself. Notice however that common issues (charset, factors and column names) are handled by `odb.read` and `odb.write`, which can deal with any SQL query.

Author(s)

Sylvain Mareschal

See Also

`odb.open`

See `ODB-package` for a complete example.

odb.close *Closes an "ODB" connection and updates the .odb file*

Description

Closes the connection to the embedded HSQLDB, removing temporary files and updating the .odb file if asked to do so.

Usage

```
odb.close(odb, write = TRUE)
```

Arguments

odb	An ODB object, as produced by odb.open.
write	Single logical value, defining if the modifications has to be saved in the original .odb file or not. Notice they will be definitively lost if closing with write=FALSE.

Value

Invisibly returns TRUE if succeeds, raises an error if not.

Note

To take data in a first file and save it in an other, copy the file before using odb.open.

Author(s)

Sylvain Mareschal

See Also

odb.open

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)

# New table
odb.write(odb, "CREATE TABLE fruits (name VARCHAR(6) PRIMARY KEY)")
odb.insert(odb, "fruits", c("banana", "pear", "peach"))

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)
```

odb.comments *Gets or sets column comments in an ODB database*

Description

This function allows comment manipulation into OpenOffice Base databases, through an odb connection.

Usage

```
odb.comments(odb, tableNames = NULL, columnNames = NULL, simplify = TRUE)
odb.comments(odb, tableNames, columnNames) <- value
```

Arguments

odb	An ODB object, as produced by odb.open.
tableNames	Character vector naming tables to search comments in. See the 'Details' section.
columnNames	Character vector naming columns to search comments in. See the 'Details' section.
simplify	Single logical value, whether to simplify returns. See the 'Value' section
value	Character vector, the comments to store.

Details

tableNames, columnNames and value can be considered as columns from a same data.frame: the first comment considered will be the comment on the first columnNames for the first tableNames, and so on. Vectors are repeated to achieve same lengths if necessary.

The NULL value can be used for tableNames and columnNames in the first syntax, with the meaning of "all possible values". See the 'Examples' section.

Value

The first syntax returns a list of character vectors. Names in the list are table names, names in vectors are column names. Only tables and columns with comments are present in the results.

With simplify set to TRUE, a character vector is directly returned (without embedding list) when a single tableNames is interrogated, and column names are skipped if a single tableNames / columnNames couple is provided.

Note

Comments on non-existing tables or columns can be manipulated as well, as there is no link between the comment storage engine and the database itself. Keep in mind these comments are stored in the .odb file, not the database itself.

Comments can also be added manually from OpenOffice while creating a table.

Author(s)

Sylvain Mareschal

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
ODB <- odb.open(odbFile)

# New tables
SQL <- c(
  "CREATE TABLE fruits (
    name VARCHAR(6) PRIMARY KEY,
    color VARCHAR(32)
    )",
  "CREATE TABLE vegetables (
    name VARCHAR(6) PRIMARY KEY,
    color VARCHAR(32)
    )"
  )
odb.write(ODB, SQL)

# Setting a single comment
odb.comments(ODB,
  tableNames = "fruits",
  columnNames = "name"
) <- "Fruit names"
print(odb.comments(ODB))

# Setting two comments in the same table
odb.comments(ODB,
  tableNames = "fruits",
  columnNames = c("name", "color")
) <- "Fruit columns"
print(odb.comments(ODB))

# Setting two distinct comments
odb.comments(ODB,
  tableNames = c("fruits", "vegetables"),
  columnNames = c("name", "color")
) <- c("Fruit names", "Vegetable colors")
print(odb.comments(ODB))

# Writes to the file and closes the connection
odb.close(ODB, write=TRUE)
```

`odb.create` *Creates a .odb empty file.*

Description

Creates an empty HSQL database embedded in a .odb file, copying a template.

Usage

```
odb.create(odbFile, template = NULL,
  overwrite = c("warning", "do", "skip", "stop"))
```

Arguments

odbFile	Path to the new file.
template	Template to copy. You should not have to care about it, as the current version is included in the package (used if NULL).
overwrite	Single character value, defining how to handle overwriting : 'warning' will overwrite and raise a warning if a file is replaced, 'stop' will raise an error before copying anything, 'do' will overwrite silently, 'skip' won't silently.

Value

Invisibly returns the single logical value returned by `file.copy`, if no critical error occurs before it is called.

Note

The default template was produced with LibreOffice Base 3.5.5.

To use a different template once, just use the `template` argument of this function. Notice a non empty database can be used as template, it will be copied and only the copy will be updated.

To update the default template, replace the "template.odb" file in the "tools" directory of the package with a new one. Usual templates are empty HSQL databases created with OpenOffice or LibreOffice, and saved as .odb files.

Notice that newer templates may require a different version of the hsql.jar library in the `odb.open` call (an error will be raised if necessary). See `odb.open` for further details on it.

Author(s)

Sylvain Mareschal

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)

# Empty template
print(odb.tables(odb))

# New table
```

```
odb.write(odb, "CREATE TABLE fruits (name VARCHAR(6) PRIMARY KEY)")
odb.insert(odb, "fruits", c("banana", "pear", "peach"))

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)


# Use as template
odbFile2 <- tempfile(fileext=".odb")
odb.create(odbFile2, template=odbFile, overwrite="do")
odb <- odb.open(odbFile2)
print(odb.tables(odb))
odb.close(odb, write=TRUE)
```

`odb.export` *Exports an ODB database to a SQL file.*

Description

Produces SQL queries describing the structure of the database and its content, and saves it to a file.

Usage

```
odb.export(odb, file)
```

Arguments

`odb`	An `ODB` object, as produced by `odb.open`.
`file`	Path to the file in which export the SQL queries.

Value

Invisibly returns TRUE if succeeds, raises an error if not.

Note

It is just a wrapper for the "SCRIPT '[file]'" SQL query, as implemented in HSQLDB.

Column comments and queries won't be exported, as they are not stored in the HSQL database.

Author(s)

Sylvain Mareschal

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)

# New table
odb.write(odb, "CREATE TABLE fruits (name VARCHAR(6) PRIMARY KEY)")
odb.insert(odb, "fruits", c("banana", "pear", "peach"))

# Export to a file
sqlFile <- tempfile(fileext=".odb")
odb.export(odb, sqlFile)

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)
```

`odb.insert` *Wrapper for inserting data in an ODB table.*

Description

Translates a data.frame into "INSERT INTO" SQL queries adaptated to a specific table from the database. Can execute the queries directly if asked to.

Usage

```
odb.insert(odb, tableName, data, execute = TRUE,
  dateFormat = "%Y-%m-%d", ...)
```

Arguments

`odb`	An `ODB` object, as produced by `odb.open`.
`tableName`	Single character value, naming the table from the database in which insert the data. No quoting is added so the HSQL engine will convert it to upper case, to refer to a case-specific table name you are required to double-quote this value.
`data`	An object coercible to data.frame, containing the data to insert into the database. Column count and order must match those of the database table, R `NA` values will be considered as SQL `NULL` values.
`execute`	Single logical value. If TRUE, the data will be inserted in the database, if FALSE the queries will be returned but not executed.
`dateFormat`	Single character value, defining how dates in "data" are formated. See the "format" argument from `strptime` for more details.
`...`	Further arguments for `odb.write`, if "execute" is set to TRUE.

Value

Returns a multiple character vector, with a distinct SQL query for each row of "data". If "execute" is set to TRUE, the return is invisible.

Author(s)

Sylvain Mareschal

See Also

odb.write

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)


## CASE-INSENSITIVE

# New table
SQL <- "CREATE TABLE fruits (
  name VARCHAR(6) PRIMARY KEY,
  color VARCHAR(32)
)"
odb.write(odb, SQL)

# Data insertion
dat <- data.frame(
  c("banana", "pear", "peach"),
  c("yellow", "yellow", "purple")
)
odb.insert(odb, "fruits", dat)

# Check content
print(odb.read(odb, "SELECT * FROM fruits"))


## CASE-SENSITIVE

# New table
SQL <- "CREATE TABLE \"Fruits\" (
  name VARCHAR(6) PRIMARY KEY,
  color VARCHAR(32)
)"
odb.write(odb, SQL)

# Data insertion
dat <- data.frame(
  c("banana", "pear", "peach"),
```

```
  c("yellow", "yellow", "purple")
)
odb.insert(odb, "\"Fruits\"", dat)

# Check content
print(odb.read(odb, "SELECT * FROM \"Fruits\""))


# Notice they are distinct tables
print(odb.tables(odb))

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)
```

---

odb.open *Creates a connection to a .odb file*

Description

Extracts embedded HSQLDB files from an .odb archive an sets a DBI connection to it, in order to read or write data from the database.

Usage

```
odb.open(odbFile, jarFile = NULL)
```

Arguments

odbFile	Path to the .odb file to connect to.
jarFile	Path to the .jar library of HSQLDB to use in JDBC. You should not have to care about it, as the current version is included in the package (used if NULL).

Value

An object of class "ODB", which will be used by every other functions of the package.

Note

The default hsql.jar library version is 1.8.0.10.

Alternate versions may be required when using .odb files not produced by odb.create, as they can embed various versions of HSQLDB. Attempting to open an .odb file with the wrong hsql.jar library version will raise an error, it is then up to the user to download the correct hsql.jar file and provide it to odb.open via the jarFile argument.

HyperSQL .jar libraries can be downloaded freely from http://www.hsqldb.org.

The default "hsql.jar" file can be replaced in the "tools" directory of the package if a different version is frequently required.

Author(s)

Sylvain Mareschal

See Also

odb.close, odb.create

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)

# New table
odb.write(odb, "CREATE TABLE fruits (name VARCHAR(6) PRIMARY KEY)")
odb.insert(odb, "fruits", c("banana", "pear", "peach"))

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)
```

---

odb.queries *Gets or sets stored queries in an ODB database*

Description

This function allows stored SQL queries manipulation into OpenOffice Base databases, through an odb connection.

Usage

```
odb.queries(odb, queryNames = NULL)
odb.queries(odb, queryNames) <- value
```

Arguments

odb	An ODB object, as produced by odb.open.
queryNames	Character vector naming queries to get or set.
value	Character vector, containing the SQL queries to store.

Value

Returns a named character vector of SQL queries.

Queries and Views

These functions manipulate OpenOffice **queries**, which are stored in the .odb file and not the database itself. They should not be confused with **views**, which are SQL features handled by the database engine.

Views are more portable (as they are stored in the database), and can be accessed as virtual tables in SQL queries. To manage them, user is required to use the SQL queries "CREATE VIEW" and "DROP VIEW" (with the `odb.write` in this package context). More informations on these SQL queries can be found in the HSQL documentation.

Note

The user is required to check and keep its queries up-to-date himself.

Queries can also be manipulated manually from OpenOffice.

Author(s)

Sylvain Mareschal

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)

# New table
odb.write(odb, "CREATE TABLE fruits (name VARCHAR(6) PRIMARY KEY)")
odb.insert(odb, "fruits", c("banana", "pear", "peach"))

# Single query
odb.queries(odb, "banana") = "SELECT * FROM fruits WHERE name='banana'"
print(odb.read(odb, odb.queries(odb, "banana")))

# Multiple queries
odb.queries(odb, c("banana","pear")) <- c(
  "SELECT * FROM fruits WHERE name='banana'",
  "SELECT * FROM fruits WHERE name='pear'"
)

# All queries
print(odb.queries(odb))

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)
```

odb.read *Executes a reading SQL query in an ODB database (SELECT ...)*

Description

Executes an SQL query expecting an output through an "odb" connection.

Usage

```
odb.read(odb, sqlQuery, stringsAsFactors = FALSE, check.names = FALSE,
  encode = TRUE, autoLogical = TRUE)
```

Arguments

odb	An ODB object, as produced by odb.open.
sqlQuery	A single character value, containing an unique SQL query to be executed.
stringsAsFactors	Single logical value : should character columns in the output be turned to factor columns or not.
check.names	Single logical value : should column names in the output be made syntactically valid by make.names or not.
encode	Single logical value : should character values be translated from UTF-8 charset (native charset for ODB files) to the locale one or not.
autoLogical	Single logical value : should logical-like columns be converted to logical or not. Notice this is a workaround, the conversion is not properly done by JDBC and the logical column type is guessed from the values.

Value

Returns a data.frame, whose content depends on the SQL query executed.

Note

To query databases built with OpenOffice or LibreOffice, it may be necessary to quote table and/or column names in sqlQuery, as the default behavior of the HSQL engine is to convert unquoted table and column names to uppercases.

Author(s)

Sylvain Mareschal

See Also

odb.write

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)

# New table
SQL <- "CREATE TABLE fruits (
  name VARCHAR(6) PRIMARY KEY,
  color VARCHAR(32)
)"
odb.write(odb, SQL)

# Data insertion
dat <- data.frame(
  c("banana", "pear", "peach"),
  c("yellow", "yellow", "purple")
)
odb.insert(odb, "fruits", dat)

# Read content
print(odb.read(odb, "SELECT * FROM fruits"))

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)
```

---

odb.tables *Gets description of every table in an ODB database.*

Description

Gets description of every tables in the database through an "odb" connection : table names, column names, column SQL and R types.

Usage

```
odb.tables(odb)
```

Arguments

odb	An ODB object, as produced by odb.open.

Value

Returns a named list, whith an element for every table in the database. Data.frames are returned by dbColumnInfo, updated with comments returned by odb.comments.

Author(s)

Sylvain Mareschal

See Also

dbColumnInfo, odb.comments

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)

# New tables
SQL <- c(
  "CREATE TABLE fruits (
    name VARCHAR(6) PRIMARY KEY,
    color VARCHAR(32)
    )",
  "CREATE TABLE vegetables (
    name VARCHAR(6) PRIMARY KEY,
    color VARCHAR(32)
    )"
  )
odb.write(odb, SQL)

# Print tables
print(odb.tables(odb))

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)
```

---

odb.write *Executes writing SQL queries in an ODB database (INSERT ...)*

Description

Executes an SQL query expecting no output through an "odb" connection.

Usage

```
odb.write(odb, sqlQueries, onError = c("warning", "stop"),
  progress = c("console", "file", "none"))
```

Arguments

odb	An ODB object, as produced by odb.open.
sqlQueries	Single or multiple character vector, with queries to be executed ordered in distinct elements. ";" query separation should not be used.
onError	Single character vector, defining how to handle SQL errors (using warning or stop). If "sqlQueries" contains more than one query, "stop" will prevent remaining queries to be executed, while "warning" won't.

progress — Single character vector, the type of progression to print when multiple queries are to be executed (See progress-class for further details).

Note

To query databases built with OpenOffice or LibreOffice, it may be necessary to quote table and/or column names in sqlQueries, as the default behavior of the HSQL engine is to convert unquoted table and column names to uppercases.

Author(s)

Sylvain Mareschal

See Also

odb.read

Examples

```
# New empty .odb file
odbFile <- tempfile(fileext=".odb")
odb.create(odbFile, overwrite="do")
odb <- odb.open(odbFile)

# New table
odb.write(odb, "CREATE TABLE fruits (name VARCHAR(6) PRIMARY KEY)")
odb.insert(odb, "fruits", c("banana", "pear", "peach"))
print(odb.tables(odb))

# Manual insert
odb.write(odb, "INSERT INTO fruits VALUES('apple')")
print(odb.read(odb, "SELECT * FROM fruits"))

# Remove table
odb.write(odb, "DROP TABLE fruits")
print(odb.tables(odb))

# Writes to the file and closes the connection
odb.close(odb, write=TRUE)
```

---

progress-class *Classes "progress", "progress.file" and "progress.console"*

Description

The two last classes inherit from the first one, and describe textual progression outputs.

Objects from the Class

`progress` is an abstract class that should not be instantiated.

`progress.file` and `progress.console` objects are to be created by the `new` function, as `initialize` methods are implemented for each of them.

See the Examples section.

progress.file constructor

Objects can be created by `new` with the following arguments :

`main`: Directly transfered in the appropriate slot

`iMax`: Directly transfered in the appropriate slot

`iCurrent`: Directly transfered in the appropriate slot

`nSteps`: Approximative amount of steps, to pass to `pretty`

progress.console constructor

Objects can be created by `new` with the following arguments :

`main`: Directly transfered in the appropriate slot

`iMax`: Directly transfered in the appropriate slot

`iCurrent`: Directly transfered in the appropriate slot

Common slots

`main`: Single character value, the title to print at beginning

`iMax`: Single integer value, the maximum value for the iteration index

`iCurrent`: Single integer value, the current value of the iteration index

progress.console slots

`pTimes`: Float vector, `proc.time` returns used to compute the ETA

`eraseLength`: Single integer value, `nchar` in the previous output

progress.file slots

`steps`: Integer vector, iteration indexes for which print an output

Methods

initialize Constructors of the classes, see previous sections.

set Updates a progression objects. Takes two arguments : `progress` (the object to update) and `iCurrent`, the new value for the iteration index. On each update, a `message` will be printed according to the class of the updated object.

Author(s)

Sylvain Mareschal

Examples

```
# File oriented progression
testFun = function() {
  obj <- new("progress.file", main="Iterating", iMax=20)
  for(i in 1:20) {
    obj = set(obj, i)
    Sys.sleep(0.1)
  }
}
testFun()

# Console oriented progression
testFun = function() {
  obj <- new("progress.console", main="Iterating", iMax=20)
  for(i in 1:20) {
    obj = set(obj, i)
    Sys.sleep(0.1)
  }
}
testFun()
```

Index