

Package ‘Rlabkey’

January 2, 2012

Version 2.1.116

Date 2011-11-29

Title Data exchange between R and LabKey Server

Author Peter Hussey

SystemRequirements curl (version 7.12.0 or higher) <<http://curl.haxx.se>>

Maintainer Cory Nathe <cnathe@labkey.com>

Description This package allows an R user to discover, query and modify data in LabKey Server

License Apache License 2.0

LazyLoad true

Depends RCurl, rjson (>= 0.1.3)

Repository CRAN

Date/Publication 2012-01-02 18:43:13

R topics documented:

| | |
|------------------------------|----|
| Rlabkey-package | 2 |
| getFolderPath | 3 |
| getLookups | 4 |
| getRows | 6 |
| getSchema | 7 |
| getSession | 8 |
| labkey.deleteRows | 10 |
| labkey.executeSql | 12 |
| labkey.getDefaultViewDetails | 14 |
| labkey.getFolders | 15 |
| labkey.getLookupDetails | 17 |
| labkey.getQueries | 18 |
| labkey.getQueryDetails | 20 |

| | |
|--------------------------------|----|
| labkey.getQueryViews | 22 |
| labkey.getSchemas | 23 |
| labkey.insertRows | 25 |
| labkey.saveBatch | 26 |
| labkey.selectRows | 28 |
| labkey.updateRows | 31 |
| lsFolders | 32 |
| lsProjects | 34 |
| lsSchemas | 35 |
| makeFilter | 36 |
| RlabkeyUsersGuide | 38 |
| saveResults | 39 |

| | |
|--------------|-----------|
| Index | 41 |
|--------------|-----------|

| | |
|-----------------|--|
| Rlabkey-package | <i>Exchange data between LabKey Server and R</i> |
|-----------------|--|

Description

This package allows the transfer of data between a LabKey Server and an R session. Data can be retrieved from LabKey into a data frame in R by specifying the query schema information (`labkey.selectRows` and `getRows`) or by using sql commands (`labkey.executeSql`). From an R session, existing data can be updated (`labkey.updateRows`), new data can be inserted (`labkey.insertRows`) or data can be deleted from the LabKey database (`labkey.deleteRows`). Interactive R users can discover available data via schema objects (`labkey.getSchema`).

Details

| | |
|-----------|--------------------|
| Package: | Rlabkey |
| Type: | Package |
| Version: | 2.1.110 |
| Date: | 2010-03-28 |
| License: | Apache License 2.0 |
| LazyLoad: | yes |

The user must have the appropriate authorization on the LabKey Server in order to modify the database through the use of these functions. Using this package to access a password protected LabKey data base requires that the user has their login information in a netrc file. The netrc file contains configuration and autologin information for the File Transfer Protocol client (ftp) and other programs such as CURL.

On a UNIX system this file should be named `.netrc` (dot netrc) and on windows it could be named `_netrc` (underscore netrc). The file should be located in the users home directory and the permissions on the file should be unreadable for everybody except the owner.

To create the `_netrc` on a windows machine, first create an environment variable called 'HOME'

that is set to your home directory (c:/Users/<User-Name> on Vista) or any directory you want to use. In that directory, create a text file named `_netrc` (note that it's underscore netrc, not dot netrc like it is on UNIX).

The following three lines must be included in the `.netrc` or `_netrc` file either separated by white space (spaces, tabs, or newlines) or commas.

```
machine <remote-machine-name>
login <user-email>
password <user-password>
```

One example would be:

```
machine localhost
login peter@labkey.com
password mypassword
```

Another example would be:

```
machine atlas.scharp.org login vobencha@fhcrc.org password mypassword
```

Multiple such blocks can exist in one file.

Author(s)

Valerie Obenchain

References

<http://www.omegahat.org/RCurl/>,
<http://dssm.unipa.it/CRAN/web/packages/rjson/rjson.pdf>,
<http://www.labkey.org/project/home/begin.view>

See Also

[labkey.selectRows](#), [labkey.executeSql](#), [makeFilter](#), [labkey.insertRows](#),
[labkey.updateRows](#), [labkey.deleteRows](#)

The Rlabkey Users Guide is available by typing `RlabkeyUsersGuide()`.

getFolderPath

Returns the folder path associated with a session

Description

Returns the current folder path for a Labkey session

Usage

```
getFolderPath(session)
```

Arguments

session the session key returned from getSession

Details

Returns a string containing the current folder path for the passed in Labkey session

Value

A character array containing the folder path, relative to the root.

Author(s)

Peter Hussey

References

<https://www.labkey.org/wiki/home/Documentation/page.view?name=projects>

See Also

[getSession lsFolders](#)

Examples

```
## Not run:  
# library(Rlabkey)  
  
lks<- getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")  
getFolderPath(lks) #returns "/apisamples"  
  
## End(Not run)
```

| | |
|------------|---|
| getLookups | <i>Get related data fields that are available to include in a query on a given query object</i> |
|------------|---|

Description

Retrieve a related query object referenced by a lookup column in the current query

Usage

```
getLookups(session, lookupField)
```

Arguments

| | |
|--------------------------|---|
| <code>session</code> | the session key returned from <code>getSession</code> |
| <code>lookupField</code> | an object representing a lookup field on LabKey Server, a named member of a query object. |

Details

Lookup fields in LabKey Server are the equivalent of declared foreign keys

Value

A query object representing the related data set. The fields of a lookup query object are usually added to the `colSelect` parameter in `getRows`. If a lookup query object is used as the query parameter in `getRows`, the call will return all of the base query columns and all of the lookup query columns. A lookup query object is very similar to base query objects that are named elements of a schema object. A lookup query object, however, does not have a parent schema object, it is only returned by `getLookups`. Also, the field names in a lookup query object are compound names relative to the base query object used in `getLookups`.

Author(s)

Peter Hussey

References

<https://www.labkey.org/wiki/home/Documentation/page.view?name=propertyFields>

See Also

[getSession](#), [getRows](#) [getSchema](#)

Examples

```
## Not run:

## get fields from lookup tables and add to query

# library(Rlabkey)
s<- getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")

scobj <- getSchema(s, "lists")

lucols <- getLookups(s, scobj$AllTypes$Category) # can add fields from related queries
lucols2 <- getLookups(s, lucols[["Category/Group"]]) # keep going to other tables

cols <- c(names(scobj$AllTypes)[2:6], names(lucols)[2:4])

getRows(s, scobj$AllTypes, colSelect=paste(cols, sep=","))
```

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

| | |
|---------|---|
| getRows | <i>Retrieve data from LabKey Server</i> |
|---------|---|

Description

Retrive rows from a LabKey Server given a session and query object

Usage

```
getRows(session, query, maxRows=NULL, colNameOpt='fieldname', ...)
```

Arguments

| | |
|------------|---|
| session | the session key returned from getSession |
| query | an object representing a query on LabKey Server, a child object of the object returned by getSchema() |
| maxRows | (optional) an integer specifying how many rows of data to return. If no value is specified, all rows are returned. |
| colNameOpt | (optional) controls the name source for the columns of the output dataframe, with valid values of 'caption', 'fieldname', and 'rname' |
| ... | Any of the remaining options to link{labkey.selectRows} |

Details

This function works as a shortcut wrapper to [labkey.selectRows](#). All of the arguments are the same as documented in [labkey.selectRows](#).

See [labkey.selectRows](#) for a discussion of the valid options and defaults for colNameOpt. Note in particular that with getRows the default is 'fieldname' instead of 'caption'.

Value

A data frame containing the query results corresponding to the default view of the specified query.

Author(s)

Peter Hussey

See Also

[getSession](#), [getSchema](#), [getLookups](#), [saveResults](#) [labkey.selectRows](#)

Examples

```
## Not run:

## simple example of getting data using schema objects

# library(Rlabkey)
s<-getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")
s # shows schemas

scobj <- getSchema(s, "lists")
scobj # shows available queries

scobj$AllTypes ## this is the query object

getRows(s, scobj$AllTypes)

## End(Not run)
```

| | |
|-----------|---|
| getSchema | <i>Returns an object representing a Labkey schema</i> |
|-----------|---|

Description

Creates and returns an object representing a Labkey schema, containing child objects representing Labkey queries

Usage

```
getSchema(session, schemaIndex)
```

Arguments

| | |
|-------------|---|
| session | the session key returned from getSession |
| schemaIndex | the name of the schema that contains the table on which you want to base a query, or the number of that schema as displayed by print(session) |

Details

Creates and returns an object representing a Labkey schema, containing child objects representing Labkey queries. This compound object is created by calling `labkey.getQueries` on the requested schema and `labkey.getQueryDetails` on each returned query. The information returned in the schema objects is essentially the same as the schema and query objects shown in the Schema Browser on LabKey Server.

Value

an object representing the schema. The named elements of a schema are the queries within that schema.

Author(s)

Peter Hussey

References

<https://www.labkey.org/wiki/home/Documentation/page.view?name=querySchemaBrowser>

See Also

[getSession](#)

Examples

```
## Not run:

## the basics of using session, schema, and query objects

# library(Rlabkey)

s<- getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")

sch<- getSchema(s, "lists")

sch$AllTypes$Category      # can walk down the populated schema tree from schema node or query node
sch$AllTypes$Category$caption
sch$AllTypes$Category$type

lucols <- getLookups(s, sch$AllTypes$Category) # can add fields from related queries

cols <- c(names(sch$AllTypes[2:6]), names(lucols)[2:4])

getRows(s, sch$AllTypes, colSelect=cols)

## End(Not run)
```

getSession

Creates and returns a LabKey Server session

Description

The session object holds server address and folder context for a user working with LabKey Server. The session-based model supports more efficient interactive use of LabKey Server from R.

Usage

```
getSession(baseUrl, folderPath="/home", curlOptions=NULL, lkOptions=NULL)
```

Arguments

| | |
|-------------|--|
| baseUrl | a string specifying the address of the LabKey Server, including the context root |
| folderPath | a string specifying the hierarchy of folders to the current folder (container) for the operation, starting with the project folder |
| curlOptions | (optional) a list of curlOptions to be set on connections to the LabKey Server, see details |
| lkOptions | (optional) a list of settings for default behavior on naming of objects, see details |

Details

Creates a session key that is passed to all the other session-based functions. Associated with the key are a baseUrl and a folderPath which determine the security context.

curlOptions

The curlOptions parameter gives a mechanism to pass control options down to the RCurl library used by Rlabkey. This can be very useful for debugging problems or setting proxy server properties. See example for debugging.

lkOptions

The lkOptions parameter gives a mechanism to change default behavior in the objects returned by Rlabkey. Currently the only available options are colNameOpt, which affects the names of columns in the data frames returned by getRows(), and maxRows, which sets a default value for this parameter when calling getRows()

Value

getSession returns a session object that represents a specific user within a specific project folder within the LabKey Server identified by the baseUrl. The combination of user, server and project/folder determines the security context of the client program. See the Rlabkey Users Guide for more discussion of how the user identity is established.

Author(s)

Peter Hussey

See Also

[getRows](#), [getSchema](#), [getLookups](#) [saveResults](#)

The Rlabkey Users Guide is available by typing RlabkeyUsersGuide().

Examples

```
## Not run:

# library(Rlabkey)
s<-getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")
s #shows schemas
```

```

scobj <- getSchema(s, "lists")
scobj  #shows available queries

scobj$AllTypes  #this is the query object

lkdata<- getRows(s, scobj$AllTypes) #shorthand for labkey.selectRows, all the same args apply
lkdata

lucols <- getLookups(s, scobj$AllTypes$Category)  #can add fields from related queries
lucols

lucols2 <- getLookups(s, lucols[["Category/Group"]]) # keep going to other tables

cols <- c(names(scobj$AllTypes)[2:6], names(lucols)[2:4])

getRows(s, scobj$AllTypes, colSelect=paste(cols, sep=","))

## using lkOptions
## change the default column naming to be the same as used in the default labkey.data data frame in R views
## with rname, spaces and slashes are replace with underscores, and the whole name is lower cased

lkOptions<-list("colNameOpt"="rname")
srname <-getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples", lkOptions=lkOptions)
getRows(srname, scobj$AllTypes)

## using the curlOptions for generating debug tracesof network traffic
d<- debugGatherer()
copt <- curlOptions(debugfunction=d$update, verbose=TRUE, cookiefile='/cooks.txt')
sdbg<- getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples",
  curlOptions=copt)
getRows(sdbg, scobj$AllTypes)
strwrap(d$value(), 100)

## End(Not run)

```

labkey.deleteRows *Delete rows of data from a LabKey database*

Description

Specify rows of data to be deleted from the LabKey Server

Usage

```
labkey.deleteRows(baseUrl, folderPath, schemaName, queryName, toDelete)
```

Arguments

| | |
|------------|---|
| baseUrl | a string specifying the baseUrl for LabKey server |
| folderPath | a string specifying the folderPath |
| schemaName | a string specifying the schemaName for the query |
| queryName | a string specifying the queryName |
| toDelete | a data frame containing a single column of data containing the data identifiers of the rows to be deleted |

Details

A single row or multiple rows of data can be deleted. For the toDelete data frame, version 0.0.5 or later accepts either a single column of data containing the data identifiers (e.g., key or lsid) or the entire row of data to be deleted. The names of the data in the data frame must be the column names from the LabKey Server. The data frame must be created with the stringsAsFactors set to FALSE.

NOTE: Each variable in a dataset has both a column label and a column name. The column label is visible at the top of each column on the web page and is longer and more descriptive. The column name is shorter and is used “behind the scenes” for database manipulation. It is the column name that must be used in the Rlabkey functions when a column name is expected. To identify a particular column name in a dataset on a web site, use the “export to R script” option available as a drop down option under the “views” tab for each dataset.

In versions 0.0.5 and earlier, labkey.deleteRows had a stripAllHidden argument. This argument did not perform a useful function and has since been removed.

Value

A list is returned with named categories of **command**, **rowsAffected**, **rows**, **queryName**, **containerPath** and **schemaName**. The **schemaName**, **queryName** and **containerPath** properties contain the same schema, query and folder path used in the request. The **rowsAffected** property indicates the number of rows affected by the API action. This will typically be the same number as passed in the request. The **rows** property contains a list of rows corresponding to the rows deleted.

Author(s)

Valerie Obenchain

See Also

[labkey.selectRows](#), [labkey.executeSql](#), [makeFilter](#), [labkey.insertRows](#),
[labkey.updateRows](#)

Examples

```
## Not run:

## Insert, update and delete
## Note that users must have the necessary permissions in the LabKey Server
```

```

## to be able to modify data through the use of these functions
# library(Rlabkey)

newrow <- data.frame(
  DisplayFld="Inserted from R"
  , TextFld="how its done"
  , IntFld= 98
  , DoubleFld = 12.345
  , DateTimeFld = "03/01/2010"
  , BooleanFld= FALSE
  , LongTextFld = "Four score and seven years ago"
  # , AttachmentFld = NA #attachment fields not supported
  , RequiredText = "Veni, vidi, vici"
  , RequiredInt = 0
  , Category = "LOOKUP2"
  , stringsAsFactors=FALSE)

insertedRow <- labkey.insertRows("http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists", que
newRowId <- insertedRow$rows[[1]]$RowId

selectedRow<-labkey.selectRows("http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists", que
, colFilter=makeFilter(c("RowId", "EQUALS", newRowId)))
selectedRow

updaterow=data.frame(
  RowId=newRowId
  , DisplayFld="Updated from R"
  , TextFld="how to update"
  , IntFld= 777
  , stringsAsFactors=FALSE)

updatedRow <- labkey.updateRows("http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists", que
selectedRow<-labkey.selectRows("http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists", que
, colFilter=makeFilter(c("RowId", "EQUALS", newRowId)))
selectedRow

deleterow <- data.frame(RowId=newRowId, stringsAsFactors=FALSE)
result <- labkey.deleteRows(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists"
result

## End(Not run)

```

labkey.executeSql

Retrieve data from a LabKey Server using SQL commands

Description

Use Sql commands to specify data to be imported into R. Prior to import, data can be manipulated through standard SQL commands supported in LabKey SQL.

Usage

```
labkey.executeSql(baseUrl, folderPath, schemaName, sql, maxRows = NULL,  
rowOffset = NULL, showHidden = FALSE, colNameOpt='caption')
```

Arguments

| | |
|------------|--|
| baseUrl | a string specifying the baseUrl for the labkey server |
| folderPath | a string specifying the folderPath |
| schemaName | a string specifying the schemaName for the query |
| sql | a string containing the sql commands to be executed |
| maxRows | (optional) an integer specifying the maximum number of rows to return. If no value is specified, all rows are returned. |
| rowOffset | (optional) an integer specifying which row of data should be the first row in the retrieval. If no value is specified, rows will begin at the start of the result set. |
| showHidden | (optional) a logical value indicating whether or not to return data columns that would normally be hidden from user view. If no value is specified, the hidden columns are not returned. |
| colNameOpt | (optional) controls the name source for the columns of the output dataframe, with valid values of 'caption', 'fieldname', and 'rname' See labkey.selectRows for more details. |

Details

A full dataset or any portion of a dataset can be imported into an R data frame using the `labkey.executeSql` function. Function arguments are components of the url that identify the location of the data and the SQL actions that should be taken on the data prior to import.

See [labkey.selectRows](#) for a discussion of the valid options and defaults for `colNameOpt`.

Value

The requested data are returned in a data frame with `stringsAsFactors` set to `FALSE`. Column names are set as determined by the `colNameOpt` parameter.

Author(s)

Valerie Obenchain

See Also

[labkey.selectRows](#), [makeFilter](#), [labkey.insertRows](#), [labkey.updateRows](#),
[labkey.deleteRows](#), [getRows](#)

Examples

```
## Not run:

## Example of an explicit join and use of group by and aggregates
# library(Rlabkey)

sql<- "SELECT AllTypesCategories.Category AS Category,
SUM(AllTypes.IntFld) AS SumOfIntFld,
AVG(AllTypes.DoubleFld) AS AvgOfDoubleFld

FROM AllTypes LEFT JOIN AllTypesCategories ON (AllTypes.Category = AllTypesCategories.TextKey)
WHERE AllTypes.Category IS NOT NULL
GROUP BY AllTypesCategories.Category"

sqlResults <- labkey.executeSql(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  sql = sql)

sqlResults

## End(Not run)
```

```
labkey.getDefaultViewDetails
```

Retrieve the fields of a LabKey query view

Description

Fetch a list of output fields and their attributes that are available from the default view of a given query

Usage

```
labkey.getDefaultViewDetails(baseUrl, folderPath, schemaName, queryName)
```

Arguments

| | |
|------------|---|
| baseUrl | a string specifying the baseUrl for the labkey server |
| folderPath | a string specifying the folderPath |
| schemaName | a string specifying the schemaName for the query |
| queryName | a string specifying the queryName |

Details

Queries have a default “views” associated with them. A query view can describe a subset or superset of the fields defined by the query. A query view is defined by using the “Customize View” button option on a LabKey data grid page. `getDefaultViewDetails` has the same arguments and returns the same shape of result data frame as `getQueryDetails`. The default view is the what you will get back on calling `labkey.selectRows` or `getRows`.

Value

The output field attributes of the default view are returned as a data frame. See [labkey.getQueryDetails](#) for a description.

Author(s)

Peter Hussey, peter@labkey.com

See Also

Retrieve data: [labkey.selectRows](#), [makeFilter](#), [labkey.executeSql](#)

Modify data: [labkey.updateRows](#), [labkey.insertRows](#), [labkey.deleteRows](#)

List available data: [labkey.getSchemas](#), [labkey.getQueries](#), [labkey.getQueryViews](#), [labkey.getQueryDetails](#), [labkey.getLookupDetails](#)

Examples

```
## Not run:

## Details of fields of a default query view
# library(Rlabkey)

queryDF <- labkey.getDefaultViewDetails(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes")

queryDF

## End(Not run)
```

`labkey.getFolders`

Retrieve a list of folders accessible to the current user

Description

Fetch a list of all folders accessible to the current user, starting from a given folder.

Usage

```
labkey.getFolders(baseUrl, folderPath, includeSubfolders=FALSE, depth=50)
```

Arguments

| | |
|-------------------|--|
| baseUrl | a string specifying the address of the LabKey Server, including the context root |
| folderPath | the starting point for the search. |
| includeSubfolders | whether the search for subfolders should recurse down the folder hierarchy |
| depth | maximum number of subfolder levels to show if includeSubfolders=TRUE |

Details

Folders are a hierarchy of containers for data and files. They are the place where permissions are set in LabKey Server. The top level in a folder hierarchy is the project. Below the project is an arbitrary hierarchy of folders that can be used to partition data for reasons of security, visibility, and organization.

Folders cut across schemas. Some schemas, like the lists schema are not visible in a folder that has no list objects defined in it. Other schemas are visible in all folders.

Value

The available schemas are returned as a single-column data frame..

Author(s)

Peter Hussey, peter@labkey.com

See Also

Retrieve data: [labkey.selectRows](#), [makeFilter](#), [labkey.executeSql](#)
Modify data: [labkey.updateRows](#), [labkey.insertRows](#), [labkey.deleteRows](#)
List available data: [labkey.getQueries](#), [labkey.getQueryViews](#), [labkey.getQueryDetails](#),
[labkey.getDefaultViewDetails](#), [labkey.getLookupDetails](#),

Examples

```
## Not run:  
  
## List of folders  
# library(Rlabkey)  
folders <- labkey.getFolders(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")  
folders  
  
## End(Not run)
```

`labkey.getLookupDetails`*Retrieve detailed information on a LabKey query*

Description

Fetch a list of output columns and their attributes from the query referenced by a lookup field

Usage

```
labkey.getLookupDetails(baseUrl, folderPath, schemaName, queryName, lookupKey)
```

Arguments

| | |
|-------------------------|--|
| <code>baseUrl</code> | a string specifying the address of the LabKey Server, including the context root |
| <code>folderPath</code> | a string specifying the hierarchy of folders to the current folder (container) for the operation, starting with the project folder |
| <code>schemaName</code> | a string specifying the schema name in which the query object is defined |
| <code>queryName</code> | a string specifying the name the query |
| <code>lookupKey</code> | a string specifying the qualified name of a lookup field (foreign key) relative to the query specified by <code>queryName</code> |

Details

When `getQueryDetails` returns non-NA values for the `lookupQueryName`, the `getLookupDetails` function can be called to enumerate the fields from the query referenced by the lookup. These lookup fields can be added to the `colSelect` list of `selectRows`.

Value

The available schemas are returned as a data frame, with the same columns as detailed in [labkey.getQueryDetails](#)

Author(s)

Peter Hussey, peter@labkey.com

See Also

Retrieve data: [labkey.selectRows](#), [makeFilter](#), [labkey.executeSql](#)

Modify data: [labkey.updateRows](#), [labkey.insertRows](#), [labkey.deleteRows](#)

List available data: [labkey.getSchemas](#), [labkey.getQueries](#), [labkey.getQueryViews](#), [labkey.getQueryDetails](#), [labkey.getDefaultViewDetails](#)

Examples

```
## Not run:
```

```
## Details of fields of a query referenced by a lookup field
# library(Rlabkey)
```

```
lu1 <- labkey.getLookupDetails(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes",
  lookupKey="Category"
)
lu1
```

```
## When a lookup field points to a query object that itself has a lookup field, use a compound fieldkey
## consisting of the lookup fields from the base query object to the target lookupDetails, separated by forward slashes
lu2<- labkey.getLookupDetails(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes",
  lookupKey="Category/Group"
)
lu2
```

```
## Now select a result set containing a field from the base query, a field from the 1st level of lookup, and one from
rows<- labkey.selectRows(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes",
  colSelect=c("DisplayFld", "Category/Category", "Category/Group/GroupName"),
  colFilter = makeFilter(c("Category/Group/GroupName", "NOT_EQUALS", "TypeRange"))
  , maxRows=20
)
rows
```

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

labkey.getQueries

Retrieve a list of available queries for a specified LabKey schema

Description

Fetch a list of queries available to the current user within in a specified folder context and specified schema

Usage

```
labkey.getQueries(baseUrl, folderPath, schemaName)
```

Arguments

| | |
|------------|--|
| baseUrl | a string specifying the address of the LabKey Server, including the context root |
| folderPath | a string specifying the hierarchy of folders to the current folder (container) for the operation, starting with the project folder |
| schemaName | a string specifying the schema name in which the query object is defined |

Details

“Query” is the LabKey term for a data container that acts like a relational table within LabKey Server. Queries include lists, assay data results, user-defined queries, built-in SQL tables in individual modules, and tables or table-like objects in external schemas. For a specific queryable object, the data that is visible depends on the current user’s permissions in a given folder. Function arguments identify the location of the server and the folder path.

Value

The available queries are returned as a three-column data frame containing one row for each field for each query in the specified schema. The three columns are

queryName the name of the query object, repeated once for every field defined as output of the query.
fieldName the name of a query output field
caption the caption of the named field as shown in the column header of a data grid, also known as a label

Author(s)

Peter Hussey, peter@labkey.com

References

<http://www.omegahat.org/RCurl/>,
<http://dssm.unipa.it/CRAN/web/packages/rjson/rjson.pdf>,
<https://www.labkey.org/project/home/begin.view>

See Also

Retrieve data: [labkey.selectRows](#), [makeFilter](#), [labkey.executeSql](#)
Modify data: [labkey.updateRows](#), [labkey.insertRows](#), [labkey.deleteRows](#)
List available data: [labkey.getSchemas](#), [labkey.getQueryViews](#), [labkey.getQueryDetails](#),
[labkey.getDefaultViewDetails](#), [labkey.getLookupDetails](#)

Examples

```
## Not run:
## List of queries in a schema
# library(Rlabkey)
queriesDF <- labkey.getQueries(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists"
)

## End(Not run)
```

```
labkey.getQueryDetails
```

Retrieve detailed information on a LabKey query

Description

Fetch a list of output columns and their attributes that are available from a given query

Usage

```
labkey.getQueryDetails(baseUrl, folderPath, schemaName, queryName)
```

Arguments

| | |
|------------|--|
| baseUrl | a string specifying the address of the LabKey Server, including the context root |
| folderPath | a string specifying the hierarchy of folders to the current folder (container) for the operation, starting with the project folder |
| schemaName | a string specifying the schema name in which the query object is defined |
| queryName | a string specifying the name of the query |

Details

Queries have a default output list of fields defined by the "default view" of the query. To retrieve that set of fields with their detailed properties such as type and nullability, use `labkey.getQueryDetails` function. Function arguments are the components of the url that identify the location of the server, the folder path, the schema, and the name of the query.

The results from `getQueryDetails` describe the "field names" that are used to build the `colSelect`, `colFilter` and `colSort` parameters to `selectRows`. Each column in the data frame returned from `selectRows` corresponds to a field in the `colSelect` list.

There are two types of fieldNames that will be reported by the server in the output of this function. For fields that are directly defined in the query corresponding the `queryName` parameter for this function, the `fieldName` is simply the name assigned by the query. Because `selectRows` returns the results specified by the default view, however, there may be cases where this default view incorporates data from other queries that have a defined 1-M relationship with the table designated by

the queryName. Such fields in related tables are referred to as “lookup” fields. Lookup fields have multi-part names using a forward slash as the delimiter. For example, in a samples data set, if the ParticipantId identifies the source of the sample, ParticipantId/CohortId/CohortName could be a reference to a CohortName field in a Cohorts data set.

These lookup fieldNames can appear in the default view and show up in the selectRows result. If a field from a lookup table is not in the default view, it can still be added to the output column list of labkey.selectRows. Use the labkey.getLookups to discover what additional fields are available via lookups, and then put their multipart fieldName values into the colSelect list. Lookup fields have the semantics of a LEFT JOIN in SQL, such that every record from the target queryName appears in the output whether or not there is a matching lookup field value.

Value

The available schemas are returned as a data frame,

queryName the name of the query, repeated n times, where n is the number of output fields from the query

fieldName the fully qualified name of the field, relative to the specified queryName.

caption a more readable label for the data field, appears as a column header in grids

fieldKey the name part that identifies this field within its containing table, independent of its use as a lookup target.

type a string specifying the field type, e.g. Text, Number, Date, Integer

isNullable TRUE if the field can be left empty (null)

isKeyField TRUE if the field is part of the primary key

isAutoIncrement TRUE if the system will automatically assign a sequential integer in this on inserting a record

isVersionField TRUE if the field is used to detect changes since last read

isHidden TRUE if the field is not displayed by default

isSelectable reserved for future use.

isUserEditable reserved for future use.

isReadOnly reserved for future use

isMvEnabled reserved for future use

lookupKeyField for a field defined as a lookup the primary key column of the query referenced by the lookup field; NA for non-lookup fields

lookupSchemaName the schema of the query referenced by the lookup field; NA for non-lookup fields

lookupDisplayField the field from the query referenced by the lookup field that is shown by default in place of the lookup field; NA for non-lookup fields

lookupQueryName the query referenced by the lookup field; NA for non-lookup fields. A non-NA value indicates that you can use this field in a call to getLookups

lookupIsPublic reserved for future use

Author(s)

Peter Hussey, peter@labkey.com

See Also

Retrieve data: [labkey.selectRows](#), [makeFilter](#), [labkey.executeSql](#)

Modify data: [labkey.updateRows](#), [labkey.insertRows](#), [labkey.deleteRows](#)

List available data: [labkey.getSchemas](#), [labkey.getQueries](#), [labkey.getQueryViews](#), [labkey.getDefaultViewDetails](#), [labkey.getLookupDetails](#)

Examples

```
## Not run:

## Details of fields of a query
# library(Rlabkey)

queryDF<-labkey.getQueryDetails(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes")

## End(Not run)
```

`labkey.getQueryViews` *Retrieve a list of available named views defined on a query in a schema*

Description

Fetch a list of named query views available to the current user in a specified folder context, schema and query

Usage

```
labkey.getQueryViews(baseUrl, folderPath, schemaName, queryName)
```

Arguments

| | |
|-------------------------|--|
| <code>baseUrl</code> | a string specifying the address of the LabKey Server, including the context root |
| <code>folderPath</code> | a string specifying the hierarchy of folders to the current folder (container) for the operation, starting with the project folder |
| <code>schemaName</code> | a string specifying the schema name in which the query object is defined |
| <code>queryName</code> | a string specifying the name the query |

Details

Queries have a default “view” associated with them, and can also have any number of named views. A named query view is created by using the “Customize View” button option on a LabKey data grid page. Use `getDefaultViewDetails` to get information about the default (unnamed) view.

Value

The available views for a query are returned as a three-column data frame, with one row per view output field.

viewName The name of the view, or NA for the default view.

fieldName The name of a field within the view, as defined in the query object to which the field belongs

key The name of the field relative to the base query, Use this value in the colSelect parameter of labkey.selectRows() .

Author(s)

Peter Hussey, peter@labkey.com

References

<https://www.labkey.org/wiki/home/Documentation/page.view?name=savingViews>

See Also

Retrieve data: [labkey.selectRows](#), [makeFilter](#), [labkey.executeSql](#)

Modify data: [labkey.updateRows](#), [labkey.insertRows](#), [labkey.deleteRows](#)

List available data: [labkey.getSchemas](#), [labkey.getQueries](#), [labkey.getQueryDetails](#), [labkey.getDefaultViewDetails](#)

Examples

```
## Not run:

## List of views defined for a query in a schema
# library(Rlabkey)

viewsDF <- labkey.getQueryViews(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes"
)

## End(Not run)
```

labkey.getSchemas

Retrieve a list of available schemas from a labkey database

Description

Fetch a list of schemas available to the current user in a specified folder context

Usage

```
labkey.getSchemas(baseUrl, folderPath)
```

Arguments

| | |
|------------|--|
| baseUrl | a string specifying the address of the LabKey Server, including the context root |
| folderPath | a string specifying the hierarchy of folders to the current folder (container) for the operation, starting with the project folder |

Details

Schemas act as the name space for query objects in LabKey Server. Schemas are generally associated with a Labkey Server "module" that provides some specific functionality. Within a queryable object, the specific data that is visible depends on the current user's permissions in a given folder. Function arguments are the components of the url that identify the location of the server and the folder path.

Value

The available schemas are returned as a single-column data frame..

Author(s)

Peter Hussey, peter@labkey.com

References

<http://www.omegahat.org/RCurl/>,
<http://dssm.unipa.it/CRAN/web/packages/rjson/rjson.pdf>,
<https://www.labkey.org/project/home/begin.view>

See Also

Retrieve data: [labkey.selectRows](#), [makeFilter](#), [labkey.executeSql](#)
Modify data: [labkey.updateRows](#), [labkey.insertRows](#), [labkey.deleteRows](#)
List available data: [labkey.getQueries](#), [labkey.getQueryViews](#), [labkey.getQueryDetails](#),
[labkey.getDefaultViewDetails](#), [labkey.getLookupDetails](#),

Examples

```
## Not run:  
  
## List of schemas  
# library(Rlabkey)  
  
schemasDF <- labkey.getSchemas(  
  baseUrl="http://localhost:8080/labkey",  
  folderPath="/apisamples"  
)
```

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

```
labkey.insertRows      Insert new rows of data into a LabKey Server
```

Description

Insert new rows of data into the database.

Usage

```
labkey.insertRows(baseUrl, folderPath, schemaName, queryName, toInsert)
```

Arguments

| | |
|------------|---|
| baseUrl | a string specifying the baseUrl for the labkey server |
| folderPath | a string specifying the folderPath |
| schemaName | a string specifying the schemaName for the query |
| queryName | a string specifying the queryName |
| toInsert | a data frame containing rows of data to be inserted |

Details

A single row or multiple rows of data can be inserted. The `toInsert` data frame must contain values for each column in the dataset and must be created with the `stringsAsFactors` option set to `FALSE`. The names of the data in the data frame must be the column names from the LabKey Server. To insert a value of `NULL`, use an empty string (`""`) in the data frame (regardless of the database column type). Also, when inserting data into a study dataset, the sequence number must be specified..

Value

A list is returned with named categories of **command**, **rowsAffected**, **rows**, **queryName**, **containerPath** and **schemaName**. The **schemaName**, **queryName** and **containerPath** properties contain the same schema, query and folder path used in the request. The **rowsAffected** property indicates the number of rows affected by the API action. This will typically be the same number as passed in the request. The **rows** property contains a list of row objects corresponding to the rows inserted.

Author(s)

Valerie Obenchain

See Also

[labkey.selectRows](#), [labkey.executeSql](#), [makeFilter](#), [labkey.updateRows](#),
[labkey.deleteRows](#)

Examples

```

## Not run:

## Insert, update and delete
## Note that users must have the necessary permissions in the database
## to be able to modify data through the use of these functions

# library(Rlabkey)

newrow <- data.frame(
  DisplayFld="Inserted from R"
  , TextFld="how its done"
  , IntFld= 98
  , DoubleFld = 12.345
  , DateTimeFld = "03/01/2010"
  , BooleanFld= FALSE
  , LongTextFld = "Four score and seven years ago"
  # , AttachmentFld = NA #attachment fields not supported
  , RequiredText = "Veni, vidi, vici"
  , RequiredInt = 0
  , Category = "LOOKUP2"
  , stringsAsFactors=FALSE)

insertedRow <- labkey.insertRows("http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists", que
newRowId <- insertedRow$rows[[1]]$RowId

selectedRow<-labkey.selectRows("http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists", que
  , colFilter=makeFilter(c("RowId", "EQUALS", newRowId)))
updaterow=data.frame(
  RowId=newRowId
  , DisplayFld="Updated from R"
  , TextFld="how to update"
  , IntFld= 777
  , stringsAsFactors=FALSE)

updatedRow <- labkey.updateRows("http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists", que
selectedRow<-labkey.selectRows("http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists", que
  , colFilter=makeFilter(c("RowId", "EQUALS", newRowId)))

deleterow <- data.frame(RowId=newRowId, stringsAsFactors=FALSE)
result <- labkey.deleteRows(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists"

## End(Not run)

```

labkey.saveBatch

Save an assay batch object to a labkey database

Description

Save an assay batch object to a labkey database

Usage

```
labkey.saveBatch(baseUrl, folderPath, assayName, resultDataFrame, batchPropertyList=NULL, runPropertyList=NULL)
```

Arguments

| | |
|-------------------|---|
| baseUrl | a string specifying the baseUrl for the labkey server |
| folderPath | a string specifying the folderPath |
| assayName | a string specifying the name of the assay instance |
| resultDataFrame | a data frame containing rows of data to be inserted |
| batchPropertyList | a list of batch Properties |
| runPropertyList | a list of run Properties |

Details

To save an R data.frame an assay results sets, you must create a named assay using the "General" assay provider. Detailed instructions are available in the Rlabkey Users Guide, accessible by entering `RlabkeyUsersGuide()` at the R command prompt. Note that `saveBatch` currently supports only a single run with one result set per batch.

Value

Returns the object representation of the Assay batch.

Author(s)

Peter Hussey

References

<https://www.labkey.org/wiki/home/Documentation/page.view?name=createDatasetViaAssay>

See Also

[labkey.selectRows](#), [labkey.executeSql](#), [makeFilter](#), [labkey.updateRows](#),
[labkey.deleteRows](#)

Examples

```
## Not run:  
  
## Very simple example of an analysis flow: query some data, calculate some stats,  
## then save the calculations as an assay result set in LabKey Server  
## Note this example expects to find an assay named "SimpleMeans" in the apisamples project  
# library(Rlabkey)  
simplifiedf <- labkey.selectRows(  
  baseUrl="http://localhost:8080/labkey",
```

```

folderPath="/apisamples",
schemaName="lists",
queryName="AllTypes")

## some dummy calculations to produce an example analysis result
testtable <- simpledf[,3:4]
colnames(testtable) <- c("IntFld", "DoubleFld")
row <- c(list("Measure"="colMeans"), colMeans(testtable, na.rm=TRUE))
results <- data.frame(row, row.names=NULL, stringsAsFactors=FALSE)
row <- c(list("Measure"="colSums"), colSums(testtable, na.rm=TRUE))
results <- rbind(results, as.vector(row))

bprops <- list(LabNotes="this is a simple demo")
bpl <- list(name=paste("Batch ", as.character(date())),properties=bprops)
rpl <- list(name=paste("Assay Run ", as.character(date())))

assayInfo<- labkey.saveBatch(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  "SimpleMeans",
  results,
  batchPropertyList=bpl,
  runPropertyList=rpl
)

## End(Not run)

```

labkey.selectRows *Retrieve data from a labkey database*

Description

Import full datasets or selected rows into R. The data can be sorted and filtered prior to import.

Usage

```

labkey.selectRows(baseUrl, folderPath, schemaName, queryName,
  viewName = NULL, colSelect = NULL, maxRows = NULL,
  rowOffset = NULL, colSort = NULL, colFilter = NULL, showHidden = FALSE, colNameOpt="caption", containerF

```

Arguments

| | |
|------------|---|
| baseUrl | a string specifying the baseUrl for the labkey server |
| folderPath | a string specifying the folderPath |
| schemaName | a string specifying the schemaName for the query |
| queryName | a string specifying the queryName |
| viewName | (optional) a string specifying the viewName associated with the query. If not specified, the default view determines the rowset returned. |

| | |
|------------------------------|--|
| <code>colSelect</code> | (optional) a vector of comma separated strings specifying which columns of a dataset or view to import |
| <code>maxRows</code> | (optional) an integer specifying how many rows of data to return. If no value is specified, all rows are returned. |
| <code>colSort</code> | (optional) a string including the name of the column to sort preceded by a “+” or “-” to indicate sort direction |
| <code>rowOffset</code> | (optional) an integer specifying which row of data should be the first row in the retrieval. If no value is specified, the retrieval starts with the first row. |
| <code>colFilter</code> | (optional) a vector or array object created by the <code>makeFilter</code> function which contains the column name, operator and value of the filter(s) to be applied to the retrieved data. |
| <code>showHidden</code> | (optional) a logical value indicating whether or not to return data columns that would normally be hidden from user view. If no value is specified, the hidden columns are not returned. In versions 0.0.5 and earlier, this argument was called <code>stripAllHidden</code> . The <code>stripAllHidden</code> argument performed the same function as <code>showHidden</code> but with different logic. The change was made for clarity. |
| <code>colNameOpt</code> | (optional) controls the name source for the columns of the output dataframe, with valid values of <code>'caption'</code> , <code>'fieldname'</code> , and <code>'rname'</code> |
| <code>containerFilter</code> | (optional) Specifies the containers to include in the scope of <code>selectRows</code> request. A value of <code>NULL</code> is equivalent to <code>"Current"</code> . Valid values are <ul style="list-style-type: none"> • <code>"Current"</code>: Include the current folder only • <code>"CurrentAndSubfolders"</code>: Include the current folder and all subfolders • <code>"CurrentPlusProject"</code>: Include the current folder and the project that contains it • <code>"CurrentAndParents"</code>: Include the current folder and its parent folders • <code>"CurrentPlusProjectAndShared"</code>: Include the current folder plus its project plus any shared folders • <code>"AllFolders"</code>: Include all folders for which the user has read permission |

Details

A full dataset or any portion of a dataset can be downloaded into an R data frame using the `labkey.selectRows` function. Function arguments are the components of the url that identify the location of the data and what actions should be taken on the data prior to import (ie, sorting, selecting particular columns or maximum number of rows, etc.).

Stored queries in LabKey Server have an associated default view and may have one or more named views. Views determine the column set of the return data frame. View columns can be a subset or superset of the columns of the underlying query— a subset if columns from the query are left out of the view, and a superset if lookup columns in the underlying query are used to include columns from related queries. Views can also include filter and sort properties that will make their result set different from the underlying query. If no view is specified, the columns and rows returned are determined by the default view, which may not be the same as the result rows of the underlying query. Please see the topic on Saving Views in the LabKey online documentation.

In the returned data frame, there are three different ways to have the columns named: `colNameOpt='caption'` uses the caption value, and is the default option for backward compatibility. It may be the best option

for displaying to another user, but may make scripting more difficult. `colNameOpt='fieldname'` uses the field name value, so that the data frame colnames are the same names that are used as arguments to `labkey` function calls. It is the default for the new `getRows` session-based function. `colNameOpt='rname'` transforms the field name value into valid R names by substituting an underscore for both spaces and forward slash (/) characters, and the lower casing the entire name. It is the way a data frame is passed to a script running at the LabKey server in the R View feature of the data grid. If you are writing scripts for running in an R view on the server, or if you prefer to work with legal r names in the returned grid, this option may be useful.

For backward compatibility, column names returned by `labkey.executeSql` and `labkey.selectRows` are field captions by default. The `getRows` function has the same `colNameOpt` parameter but defaults to field names instead of captions.

Value

The requested data are returned in a data frame with `stringsAsFactors` set to `FALSE`. Column names are set as determined by the `colNameOpt` parameter.

Author(s)

Valerie Obenchain

References

<https://www.labkey.org/wiki/home/Documentation/page.view?name=savingViews>

See Also

Retrieve data: [makeFilter](#), [labkey.executeSql](#)

Modify data: [labkey.updateRows](#), [labkey.insertRows](#), [labkey.deleteRows](#)

List available data: [labkey.getSchemas](#), [labkey.getQueries](#), [labkey.getQueryViews](#), [labkey.getQueryDetails](#), [labkey.getQueryData](#)

Examples

```
## Not run:

# library(Rlabkey)

## select from a list named AllTypes

rows <- labkey.selectRows(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes")

## select from a view on that list
viewrows <- labkey.selectRows(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples", schemaName="List")

## select a subset of columns
colSelect=c("TextFld", "IntFld")
```

```
subsetcols <- labkey.selectRows(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lis

## End(Not run)
```

labkey.updateRows *Update existing rows of data in a labkey database*

Description

Send data from an R session to update existing rows of data in the database.

Usage

```
labkey.updateRows(baseUrl, folderPath, schemaName, queryName, toUpdate)
```

Arguments

| | |
|------------|--|
| baseUrl | a string specifying the baseUrl for the labkey server |
| folderPath | a string specifying the folderPath |
| schemaName | a string specifying the schemaName for the query |
| queryName | a string specifying the queryName |
| toUpdate | a data frame containing the row(s) of data to be updated |

Details

A single row or multiple rows of data can be updated. The toUpdate data frame should contain the rows of data to be updated and must be created with the stringsAsFactors option set to FALSE. The names of the data in the data frame must be the column names from the labkey database. To update a row/column to a value of NULL, use an empty string ("") in the data frame (regardless of the database column type).

Value

A list is returned with named categories of **command**, **rowsAffected**, **rows**, **queryName**, **containerPath** and **schemaName**. The **schemaName**, **queryName** and **containerPath** properties contain the same schema, query and folder path used in the request. The **rowsAffected** property indicates the number of rows affected by the API action. This will typically be the same number as passed in the request. The **rows** property contains a list of row objects corresponding to the rows updated.

Author(s)

Valerie Obenchain

See Also

[labkey.selectRows](#), [labkey.executeSql](#), [makeFilter](#), [labkey.insertRows](#),
[labkey.deleteRows](#)

Examples

```

## Not run:

## Insert, update and delete
## Note that users must have the necessary permissions in the database
## to be able to modify data through the use of these functions
# library(Rlabkey)

newrow <- data.frame(
  DisplayFld="Inserted from R"
  , TextFld="how its done"
  , IntFld= 98
  , DoubleFld = 12.345
  , DateTimeFld = "03/01/2010"
  , BooleanFld= FALSE
  , LongTextFld = "Four score and seven years ago"
  # , AttachmentFld = NA      #attachment fields not supported
  , RequiredText = "Veni, vidi, vici"
  , RequiredInt = 0
  , Category = "LOOKUP2"
  , stringsAsFactors=FALSE)

insertedRow <- labkey.insertRows("http://localhost:8080/labkey", folderPath="/apisamples",schemaName="lists", qu
newRowId <- insertedRow$rows[[1]]$RowId

selectedRow<-labkey.selectRows("http://localhost:8080/labkey", folderPath="/apisamples",schemaName="lists", que
  , colFilter=makeFilter(c("RowId", "EQUALS", newRowId)))
selectedRow

updaterow=data.frame(
  RowId=newRowId
  , DisplayFld="Updated from R"
  , TextFld="how to update"
  , IntFld= 777
  , stringsAsFactors=FALSE)

updatedRow <- labkey.updateRows("http://localhost:8080/labkey", folderPath="/apisamples",schemaName="lists", que
selectedRow<-labkey.selectRows("http://localhost:8080/labkey", folderPath="/apisamples",schemaName="lists", que
  , colFilter=makeFilter(c("RowId", "EQUALS", newRowId)))
selectedRow

deleterow <- data.frame(RowId=newRowId, stringsAsFactors=FALSE)
result <- labkey.deleteRows(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples", schemaName="lists"
str(result)

## End(Not run)

```

Description

Lists the available folder paths relative to the current folder path for a Labkey session

Usage

```
lsFolders(session)
```

Arguments

session the session key returned from getSession

Details

Lists the available folder paths relative to the current folder path for a Labkey session

Value

A character array containing the available folder paths, relative to the root. These values can be set on a session using curFolder<-

Author(s)

Peter Hussey

References

<https://www.labkey.org/wiki/home/Documentation/page.view?name=projects>

See Also

[getSession](#), [lsProjects](#), [lsSchemas](#)

Examples

```
## Not run:

##get a list if projects and folders
# library(Rlabkey)

lsProjects(baseUrl="http://localhost:8080/labkey") # returns "/apisamples", "/home", ...

lks<- getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")

lsFolders(lks) #returns values "/apisamples" , "/apisamples/sub1" ,"/apisamples/sub1/child" ,"/apisamples/sub2"

## End(Not run)
```

`lsProjects`*List the projects available at a given Labkey Server address*

Description

Lists the projects available. Takes a string URL instead of a session, as it is intended for use before creating a session.

Usage

```
lsProjects(baseUrl)
```

Arguments

| | |
|----------------------|---|
| <code>baseUrl</code> | a string specifying the <code>baseUrl</code> for the Labkey Server, of the form <code>http://<server dns name>/<contextroot></code> |
|----------------------|---|

Details

Lists the available folder paths relative to the current folder path for a Labkey session.

Value

A character array containing the available folder paths, relative to the root. These values can be set on a session using `curFolder<-`

Author(s)

Peter Hussey

References

<https://www.labkey.org/project/home/begin.view>

See Also

[getSession](#), [lsProjects](#), [lsSchemas](#)

Examples

```
## Not run:

## get list of projects on server, connect a session in one project, then list the folders in that project
# library(Rlabkey)
lsProjects("http://www.labkey.org")

lkorg <- getSession("http://www.labkey.org", "/home")
lsFolders(lkorg)
```

```
lkorg <- getSession("http://www.labkey.org", "/home/Study/ListDemo")
lsSchemas(lkorg)

## End(Not run)
```

| | |
|-----------|-----------------------------------|
| lsSchemas | <i>List the available queries</i> |
|-----------|-----------------------------------|

Description

Lists the available schemas given the current folder path for a Labkey session

Usage

```
lsSchemas(session)
```

Arguments

session the session key returned from getSession

Details

Lists the available schemas given the current folder path for a Labkey session

Value

A character array containing the available schema names

Author(s)

Peter Hussey

See Also

[getSession](#), [lsFolders](#), [lsProjects](#)

Examples

```
## Not run:

## get a list of schemas available in the current session context
# library(Rlabkey)

lks<- getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")

lsSchemas(lks) #returns several schema names, e.g. "lists", "core", "MS1", etc.

## End(Not run)
```

`makeFilter`*Builds filters to be used in `labkey.selectRows` and `getRows`*

Description

This function takes inputs of column name, filter value and filter operator and returns an array of filters to be used in `labkey.selectRows` and `getRows`.

Usage

```
makeFilter(...)
```

Arguments

... Arguments in `c("colname", "operator", "value")` form, used to create a filter.

Details

These filters are applied to the data prior to import into R. The user can specify as many filters as desired. The format for specifying a filter is a vector of characters including the column name, operator and value.

colname a string specifying the name of the column to be filtered

operator a string specifying what operator should be used in the filter (see options below)

value an integer or string specifying the value the columns should be filtered on

Operator values:

EQUAL

NOT_EQUAL

NOT_EQUAL_OR_MISSING

DATE_EQUAL

DATE_NOT_EQUAL

MISSING

NOT_MISSING

GREATER_THAN

GREATER_THAN_OR_EQUAL

LESS_THAN

LESS_THAN_OR_EQUAL

CONTAINS

DOES_NOT_CONTAIN

STARTS_WITH

DOES_NOT_START_WITH

EQUALS_ONE_OF

MV_INDICATOR

NO_MV_INDICATOR

When using the `MISSING`, `NOT_MISSING`, `MV_INDICATOR`, or `NO_MV_INDICATOR` operators, an empty string should be supplied as the value. See example below.

Value

The function returns either a single string or an array of strings to be use in the `colFilter` argument of the `labkey.selectRows` function.

Author(s)

Valerie Obenchain

References

<http://www.omegahat.org/RCurl/>,
<http://dssm.unipa.it/CRAN/web/packages/rjson/rjson.pdf>,
<https://www.labkey.org/project/home/begin.view>

See Also

[labkey.selectRows](#)

Examples

```
## Not run:

# library(Rlabkey)

## Two filters, ANDed together
filter1<- makeFilter(c("TextFld","CONTAINS","h"), c("BooleanFld","EQUAL","TRUE"))

## Apply a filter in labkey.selectRows function
rows <- labkey.selectRows(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes",
  colFilter=filter1
)
rows

## Using "equals one of" operator:
filter2 <- makeFilter(c("RowId","EQUALS_ONE_OF","2;3;6"))
rows <- labkey.selectRows(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples",
  schemaName="lists",
  queryName="AllTypes",
  colFilter=filter2
)
rows

## Using "missing" operator:
filter3 <- makeFilter(c("IntFld","MISSING",""))
```

```
rows <- labkey.selectRows(  
  baseUrl="http://localhost:8080/labkey",  
  folderPath="/apisamples",  
  schemaName="lists",  
  queryName="AllTypes",  
  colFilter=filter3  
)  
rows  
  
## End(Not run)
```

RlabkeyUsersGuide *Open the Rlabkey Users Guide*

Description

Brings up the Rlabkey Users Guide.

Usage

```
RlabkeyUsersGuide(view=TRUE)
```

Arguments

view Leave as default TRUE

Details

Brings up the Rlabkey Users Guide.

Value

Path to the Users Guide pdf.

Author(s)

Peter Hussey

Examples

```
## Not run:  
  
# library(Rlabkey)  
##RlabkeyUsersGuide()  
  
## End(Not run)
```

| | |
|-------------|---|
| saveResults | <i>Returns an object representing a Labkey schema</i> |
|-------------|---|

Description

Creates and returns an object representing a Labkey schema, containing child objects representing Labkey queries

Usage

```
saveResults(session, assayName, resultDataFrame,  
batchPropertyList= list(name=paste("Batch ", as.character(date()))),  
runPropertyList= list(name=paste("Assay Run ", as.character(date())) )
```

Arguments

| | |
|-------------------|---|
| session | the session key returned from getSession |
| assayName | a string specifying the name of the assay instance |
| resultDataFrame | a data frame containing rows of data to be inserted |
| batchPropertyList | a list of batch Properties |
| runPropertyList | a list of run Properties |

Details

saveResults is a wrapper function to labkey.saveBatch with two changes: First, it uses a session object in place of the separate baseUrl and folderPath arguments. Second, it provides defaults for generating Batch and Run names based on a current timestamp.

To see the save result on LabKey server, click on the "SimpleMeans" assay in the Assay List web part.

Value

an object representing the assay.

Author(s)

Peter Hussey

References

<https://www.labkey.org/project/home/begin.view>

See Also

[getSession](#), [getSchema](#), [getLookups](#) [getRows](#)

Examples

```
## Not run:

## Very simple example of an analysis flow: query some data, calculate some stats,
## then save the calculations as an assay result set in LabKey Server

# library(Rlabkey)
s<- getSession(baseUrl="http://localhost:8080/labkey", folderPath="/apisamples")
scobj <- getSchema(s, "lists")
simplifiedf <- getRows(s, scobj$AllTypes)

## some dummy calculations to produce an example analysis result
testtable <- simplifiedf[,3:4]
colnames(testtable) <- c("IntFld", "DoubleFld")
row <- c(list("Measure"="colMeans"), colMeans(testtable, na.rm=TRUE))
results <- data.frame(row, row.names=NULL, stringsAsFactors=FALSE)
row <- c(list("Measure"="colSums"), colSums(testtable, na.rm=TRUE))
results <- rbind(results, as.vector(row))

bprops <- list(LabNotes="this is a simple demo")
bpl<- list(name=paste("Batch ", as.character(date())),properties=bprops)
rpl<- list(name=paste("Assay Run ", as.character(date())))

assayInfo<- saveResults(s, "SimpleMeans", results, batchPropertyList=bpl, runPropertyList=rpl)

## End(Not run)
```

Index

*Topic **IO**

- labkey.deleteRows, 10
- labkey.executeSql, 12
- labkey.getDefaultViewDetails, 14
- labkey.getFolders, 15
- labkey.getLookupDetails, 17
- labkey.getQueries, 18
- labkey.getQueryDetails, 20
- labkey.getQueryViews, 22
- labkey.getSchemas, 23
- labkey.insertRows, 25
- labkey.saveBatch, 26
- labkey.selectRows, 28
- labkey.updateRows, 31

*Topic **file**

- getFolderPath, 3
- getLookups, 4
- getRows, 6
- getSchema, 7
- getSession, 8
- lsFolders, 32
- lsProjects, 34
- lsSchemas, 35
- makeFilter, 36
- RlabkeyUsersGuide, 38
- saveResults, 39

*Topic **package**

- Rlabkey-package, 2

- getFolderPath, 3
- getLookups, 4, 6, 9, 40
- getRows, 5, 6, 9, 13, 30, 40
- getSchema, 5, 6, 7, 9, 40
- getSession, 4–6, 8, 8, 33–35, 40

- labkey.deleteRows, 3, 10, 13, 15–17, 19, 22–25, 27, 30, 31
- labkey.executeSql, 3, 11, 12, 15–17, 19, 22–25, 27, 30, 31

- labkey.getDefaultViewDetails, 14, 16, 17, 19, 22–24, 30
- labkey.getFolders, 15
- labkey.getLookupDetails, 15, 16, 17, 19, 22–24, 30
- labkey.getQueries, 15–17, 18, 22–24, 30
- labkey.getQueryDetails, 15–17, 19, 20, 23, 24, 30
- labkey.getQueryViews, 15–17, 19, 22, 22, 24, 30
- labkey.getSchemas, 15, 17, 19, 22, 23, 23, 30
- labkey.insertRows, 3, 11, 13, 15–17, 19, 22–24, 25, 30, 31
- labkey.saveBatch, 26
- labkey.selectRows, 3, 6, 11, 13, 15–17, 19, 22–25, 27, 28, 31, 37
- labkey.updateRows, 3, 11, 13, 15–17, 19, 22–25, 27, 30, 31
- lsFolders, 4, 32, 35
- lsProjects, 33, 34, 34, 35
- lsSchemas, 33, 34, 35
- makeFilter, 3, 11, 13, 15–17, 19, 22–25, 27, 30, 31, 36
- Rlabkey (Rlabkey-package), 2
- Rlabkey-package, 2
- RlabkeyUsersGuide, 38

- saveResults, 6, 9, 39