

# Package ‘Rlabkey’

September 14, 2021

**Version** 2.8.1

**Date** 2021-09-10

**Title** Data Exchange Between R and 'LabKey' Server

**Author** Peter Hussey

**Maintainer** Cory Nathe <cnathe@labkey.com>

**Description** The 'LabKey' client library for R makes it easy for R users to load live data from a 'LabKey' Server, <<https://www.labkey.com/>>, into the R environment for analysis, provided users have permissions to read the data. It also enables R users to insert, update, and delete records stored on a 'LabKey' Server, provided they have appropriate permissions to do so.

**License** Apache License 2.0

**Copyright** Copyright (c) 2010-2018 LabKey Corporation

**LazyLoad** true

**Depends** httr, jsonlite

**LinkingTo** Rcpp

**Imports** Rcpp (>= 0.11.0)

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2021-09-14 13:40:02 UTC

## R topics documented:

Rlabkey-package	3
getFolderPath	5
getLookups	6
getRows	7
getSchema	8
getSession	10
labkey.acceptSelfSignedCerts	11
labkey.curlOptions	12

labkey.deleteRows . . . . .	12
labkey.domain.create . . . . .	14
labkey.domain.createAndLoad . . . . .	17
labkey.domain.createConditionalFormat . . . . .	19
labkey.domain.createConditionalFormatQueryFilter . . . . .	20
labkey.domain.createDesign . . . . .	22
labkey.domain.createIndices . . . . .	23
labkey.domain.drop . . . . .	24
labkey.domain.FILTER_TYPES . . . . .	25
labkey.domain.get . . . . .	26
labkey.domain.inferFields . . . . .	27
labkey.domain.save . . . . .	28
labkey.executeSql . . . . .	29
labkey.experiment.createData . . . . .	31
labkey.experiment.createMaterial . . . . .	32
labkey.experiment.createRun . . . . .	33
labkey.experiment.SAMPLE_DERIVATION_PROTOCOL . . . . .	35
labkey.experiment.saveBatch . . . . .	36
labkey.experiment.saveRuns . . . . .	38
labkey.getBaseUrl . . . . .	39
labkey.getDefaultViewDetails . . . . .	40
labkey.getFolders . . . . .	41
labkey.getLookupDetails . . . . .	42
labkey.getModuleProperty . . . . .	44
labkey.getQueries . . . . .	45
labkey.getQueryDetails . . . . .	46
labkey.getQueryViews . . . . .	48
labkey.getRequestOptions . . . . .	50
labkey.getSchemas . . . . .	51
labkey.importRows . . . . .	52
labkey.insertRows . . . . .	53
labkey.makeRemotePath . . . . .	56
labkey.pipeline.getFileStatus . . . . .	57
labkey.pipeline.getPipelineContainer . . . . .	58
labkey.pipeline.getProtocols . . . . .	59
labkey.pipeline.startAnalysis . . . . .	61
labkey.provenance.addRecordingStep . . . . .	62
labkey.provenance.createProvenanceParams . . . . .	63
labkey.provenance.startRecording . . . . .	65
labkey.provenance.stopRecording . . . . .	67
labkey.query.import . . . . .	68
labkey.rstudio.initReport . . . . .	70
labkey.rstudio.initRStudio . . . . .	71
labkey.rstudio.initSession . . . . .	71
labkey.rstudio.isInitialized . . . . .	72
labkey.rstudio.saveReport . . . . .	72
labkey.saveBatch . . . . .	73
labkey.security.createContainer . . . . .	75

labkey.security.deleteContainer . . . . .	76
labkey.security.getContainers . . . . .	77
labkey.security.impersonateUser . . . . .	78
labkey.security.moveContainer . . . . .	79
labkey.security.stopImpersonating . . . . .	81
labkey.selectRows . . . . .	82
labkey.setCurlOptions . . . . .	85
labkey.setDebugMode . . . . .	86
labkey.setDefaults . . . . .	87
labkey.setModuleProperty . . . . .	88
labkey.transform.getRunPropertyValue . . . . .	89
labkey.transform.readRunPropertiesFile . . . . .	90
labkey.truncateTable . . . . .	91
labkey.updateRows . . . . .	92
labkey.webdav.delete . . . . .	94
labkey.webdav.downloadFolder . . . . .	95
labkey.webdav.get . . . . .	96
labkey.webdav.listDir . . . . .	98
labkey.webdav.mkDir . . . . .	99
labkey.webdav.mkDirs . . . . .	100
labkey.webdav.pathExists . . . . .	102
labkey.webdav.put . . . . .	103
labkey.whoAmI . . . . .	104
lsFolders . . . . .	105
lsProjects . . . . .	106
lsSchemas . . . . .	107
makeFilter . . . . .	108
RlabkeyUsersGuide . . . . .	110
saveResults . . . . .	111

**Index****113**

Rlabkey-package

*Exchange data between LabKey Server and R***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` and `labkey.importRows`) 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: 2.8.1  
Date: 2021-09-10  
License: Apache License 2.0  
LazyLoad: yes

The user must have appropriate authorization on the LabKey Server in order to access or modify data using these functions. All access to secure content on LabKey Server requires authentication via an api key (see `labkey.setDefaults` for more details) or a properly configured netrc file that includes the user's login information.

The netrc file is a standard mechanism for conveying configuration and autologin information to the File Transfer Protocol client (ftp) and other programs such as CURL. On a Linux or Mac system this file should be named `.netrc` (dot netrc) and on Windows it should be named `_netrc` (underscore netrc). The file should be located in the user's 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' set to your home directory (e.g., `c:/Users/<User-Name>` on recent versions of Windows) 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 Linux/Mac).

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

**See Also**

[labkey.selectRows](#), [labkey.executeSql](#), [makeFilter](#), [labkey.insertRows](#), [labkey.importRows](#),  
[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/Documentation/wiki-page.view?name=projects>

**See Also**

[getSession](#) [lsFolders](#)

**Examples**

```
# library(Rlabkey)

lks<- getSession("https://www.labkey.org", "/home")
getFolderPath(lks) #returns "/home"
```

---

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**

session	the session key returned from getSession
lookupField	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/Documentation/wiki-page.view?name=propertyFields>

**See Also**

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

**Examples**

```
## get fields from lookup tables and add to query
# library(Rlabkey)

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

scobj <- getSchema(s, "lists")

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

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

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

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

---

getRows

*Retrieve data from LabKey Server*


---

**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**

```
## 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)
```

---

getSchema

*Returns an object representing a LabKey schema*

---

**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/Documentation/wiki-page.view?name=querySchemaBrowser>

**See Also**

[getSession](#)

**Examples**

```
## the basics of using session, schema, and query objects
# library(Rlabkey)

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

sch<- getSchema(s, "lists")

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

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

```
cols <- c(names(sch$AllTypes[2:6]), names(lucols)[2:4])
getRows(s, sch$AllTypes, colSelect=cols)
```

---

getSession	<i>Creates and returns a LabKey Server session</i>
------------	--

---

### 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**

```
# library(Rlabkey)

s <- getSession("https://www.labkey.org", "/home")
s #shows schemas

## 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)
```

---

labkey.acceptSelfSignedCerts

*Convenience method to configure Rlabkey connections to accept self-signed certificates*

---

**Description**

Rlabkey uses the package RCurl to connect to the LabKey Server. This is equivalent to executing the function: `labkey.setCurlOptions(ssl_verifyhost=0, ssl_verifypeer=FALSE)`

**Usage**

```
labkey.acceptSelfSignedCerts()
```

---

labkey.curlOptions	<i>Returns the current set of Curl options that are being used in the existing session</i>
--------------------	--

---

### Description

Rlabkey uses the package RCurl to connect to the LabKey Server.

### Usage

```
labkey.curlOptions()
```

---

labkey.deleteRows	<i>Delete rows of data from a LabKey database</i>
-------------------	---

---

### Description

Specify rows of data to be deleted from the LabKey Server

### Usage

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

### 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
provenanceParams	the provenance parameter object which contains the options to include as part of a provenance recording. This is a premium feature and requires the Provenance LabKey module to function correctly, if it is not present this parameter will be ignored.

## 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.

## 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.importRows](#), [labkey.updateRows](#), [labkey.provenance.createProvenanceParams](#), [labkey.provenance.startRecording](#), [labkey.provenance.addRecordingStep](#), [labkey.provenance.stopRecording](#)

## Examples

```
## 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",
  queryName="AllTypes", toInsert=newrow)
newRowId <- insertedRow$rows[[1]]$RowId

selectedRow<-labkey.selectRows("http://localhost:8080/labkey",
  folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
  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",
  queryName="AllTypes", toUpdate=updaterow)
selectedRow<-labkey.selectRows("http://localhost:8080/labkey",
  folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
  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",
  queryName="AllTypes", toDelete=deleterow)
result

```

---

labkey.domain.create *Create a new LabKey domain*

---

### Description

Create a domain of the type specified by the domainKind and the domainDesign. A LabKey domain represents a table in a specific schema.

### Usage

```

labkey.domain.create(baseUrl=NULL, folderPath,
  domainKind=NULL, domainDesign=NULL, options=NULL,
  module=NULL, domainGroup=NULL, domainTemplate=NULL,
  createDomain=TRUE, importData=TRUE)

```

## Arguments

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
domainKind	(optional) a string specifying the type of domain to create
domainDesign	(optional) a list containing the domain design to create
options	(optional) a list containing options specific to the domain kind
module	(optional) the name of the module that contains the domain template group
domainGroup	(optional) the name of a domain template group
domainTemplate	(optional) the name of a domain template within the domain group
createDomain	(optional) when using a domain template, create the domain. Defaults to TRUE
importData	(optional) when using a domain template, import initial data associated in the template. Defaults to TRUE

## Details

When creating a domain using a domainKind parameter, the domainDesign parameter will be required. If a domain template is being used, then module, domainGroup, and domainTemplate are required.

Will create a domain of the specified domain type, valid types are

- "IntList": A list with an integer key field
- "VarList": A list with a string key field
- "StudyDatasetVisit": A dataset in a visit based study
- "StudyDatasetDate": A dataset in a date based study
- "IssueDefinition": An issue list domain
- "SampleSet": Sample set
- "DataClass": Data class

The domain design parameter describes the set of fields in the domain, see labkey.domain.createDesign for the helper function that can be used to construct this data structure. The options parameter should contain a list of attributes that are specific to the domain kind specified. The list of valid options for each domain kind are:

- IntList and VarList
  - keyName (required) : The name of the field in the domain design which identifies the key field
- StudyDatasetVisit and StudyDatasetDate
  - datasetId : Specifies a dataset ID to use, the default is to auto generate an ID
  - categoryId : Specifies an existing category ID
  - categoryName : Specifies an existing category name
  - demographics : (TRUE | FALSE) Determines whether the dataset is created as demographic

- keyPropertyName : The name of an additional key field to be used in conjunction with participantId and (visitId or date) to create unique records
- useTimeKeyField : (TRUE | FALSE) Specifies to use the time portion of the date field as an additional key
- isManagedField : (TRUE | FALSE) Specifies whether the field from keyPropertyName should be managed by LabKey.
- IssueDefinition
  - providerName : The type of issue list to create (IssueDefinition (default) or AssayRequestDefinition)
  - singularNoun : The singular name to use for items in the issue definition (defaults to issue)
  - pluralNoun : The plural name (defaults to issues)
- SampleSet
  - idCols : The columns to use when constructing the concatenated unique ID. Can be up to 3 numeric IDs which represent the zero-based position of the fields in the domain.
  - parentCol : The column to represent the parent identifier in the sample set. This is a numeric value representing the zero-based position of the field in the domain.
  - nameExpression : The name expression to use for creating unique IDs
- DataClass
  - sampleSet : The ID of the sample set if this data class is associated with a sample set.
  - nameExpression : The name expression to use for creating unique IDs

## Value

A list containing elements describing the newly created domain.

## Author(s)

Karl Lum

## See Also

[labkey.domain.get](#), [labkey.domain.inferFields](#), [labkey.domain.createDesign](#), [labkey.domain.createIndices](#), [labkey.domain.save](#), [labkey.domain.drop](#), [labkey.domain.createConditionalFormat](#), [labkey.domain.createCon](#), [labkey.domain.FILTER\\_TYPES](#)

## Examples

```
## create a data frame and infer it's fields, then create a domain design from it
library(Rlabkey)

df <- data.frame(ptid=c(1:3), age = c(10,20,30), sex = c("f", "m", "f"))
fields <- labkey.domain.inferFields(baseUrl="http://labkey/", folderPath="home", df=df)
dd <- labkey.domain.createDesign(name="test list", fields=fields)
```



```
## create a new list with an integer key field
labkey.domain.create(baseUrl="http://labkey/", folderPath="home",
    domainKind="IntList", domainDesign=dd, options=list(keyName = "ptid"))

## create a domain using a domain template
labkey.domain.create(baseUrl="http://labkey/", folderPath="home",
    domainTemplate="Priority", module="simpletest", domainGroup="todolist")
```

---

labkey.domain.createAndLoad

*Create a new LabKey domain and load data*

---

## Description

Create a domain of the type specified by the domainKind. A LabKey domain represents a table in a specific schema. Once the domain is created the data from the data frame will be imported.

## Usage

```
labkey.domain.createAndLoad(baseUrl=NULL, folderPath,
    name, description="", df, domainKind, options=NULL, schemaName=NULL)
```

## Arguments

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
name	a string specifying the name of the domain to create
description	(optional) a string specifying the domain description
df	a data frame specifying fields to infer. The data frame must have column names as well as row data to infer the type of the field from.
domainKind	a string specifying the type of domain to create
options	(optional) a list containing options specific to the domain kind
schemaName	(optional) a string specifying the schema name to import the data into

## Details

Will create a domain of the specified domain type, valid types are

- "IntList": A list with an integer key field
- "VarList": A list with a string key field
- "StudyDatasetVisit": A dataset in a visit based study
- "StudyDatasetDate": A dataset in a date based study
- "IssueDefinition": An issue list domain
- "SampleSet": Sample set

- "DataClass": Data class

The options parameter should contain a list of attributes that are specific to the domain kind specified. The list of valid options for each domain kind are:

- IntList and VarList
  - keyName (required) : The name of the field in the domain design which identifies the key field
- StudyDatasetVisit and StudyDatasetDate
  - datasetId : Specifies a dataset ID to use, the default is to auto generate an ID
  - categoryId : Specifies an existing category ID
  - categoryName : Specifies an existing category name
  - demographics : (TRUE | FALSE) Determines whether the dataset is created as demographic
  - keyPropertyName : The name of an additional key field to be used in conjunction with participantId and (visitId or date) to create unique records
  - useTimeKeyField : (TRUE | FALSE) Specifies to use the time portion of the date field as an additional key
- IssueDefinition
  - providerName : The type of issue list to create (IssueDefinition (default) or AssayRequestDefinition)
  - singularNoun : The singular name to use for items in the issue definition (defaults to issue)
  - pluralNoun : The plural name (defaults to issues)
- SampleSet
  - idCols : The columns to use when constructing the concatenated unique ID. Can be up to 3 numeric IDs which represent the zero-based position of the fields in the domain.
  - parentCol : The column to represent the parent identifier in the sample set. This is a numeric value representing the zero-based position of the field in the domain.
  - nameExpression : The name expression to use for creating unique IDs
- DataClass
  - sampleSet : The ID of the sample set if this data class is associated with a sample set.
  - nameExpression : The name expression to use for creating unique IDs

### Value

A list containing the newly uploaded data frame rows.

### Author(s)

Karl Lum

### See Also

[labkey.domain.get](#), [labkey.domain.inferFields](#), [labkey.domain.createDesign](#), [labkey.domain.createIndices](#), [labkey.domain.save](#), [labkey.domain.drop](#)

## Examples

```
library(Rlabkey)

## Prepare a data.frame
participants = c("0001", "0001", "0002", "0002", "0007", "0008")
Visit = c("V1", "V2", "V2", "V1", "V2", "V1")
IntValue = c(256:261)
dataset = data.frame("ParticipantID" = participants, Visit,
  "IntegerValue" = IntValue, check.names = FALSE)

## Create the dataset and import
labkey.domain.createAndLoad(baseUrl="http://labkey", folderPath="home",
  name="demo dataset", df=dataset, domainKind="StudyDatasetVisit")
```

---

```
labkey.domain.createConditionalFormat
  Create a conditional format data frame
```

---

## Description

Create a conditional format data frame.

## Usage

```
labkey.domain.createConditionalFormat(queryFilter, bold=FALSE, italic=FALSE,
  strikeThrough=FALSE, textColor="", backgroundColor="")
```

## Arguments

queryFilter	a string specifying what logical filter should be applied
bold	a boolean for if the text display should be formatted in bold
italic	a boolean for if the text display should be formatted in italic
strikeThrough	a boolean for if the text display should be formatted with a strikethrough
textColor	a string specifying the hex code of the text color for display
backgroundColor	a string specifying the hex code of the text background color for display

## Details

This function can be used to construct a conditional format data frame intended for use within a domain design's conditionalFormats component while creating or updating a domain. The queryFilter parameter can be used in conjunction with labkey.domain.createConditionalFormatQueryFilter for convenient construction of a query filter string. Multiple conditional formats can be applied to one field, where each format specified constitutes a new row of the field's conditionalFormats data frame. If text formatting options are not specified, the default is to display the value as black text on a white background.

**Value**

The data frame containing values describing a conditional format.

**Author(s)**

Rosaline Pyktel

**See Also**

[labkey.domain.get](#), [labkey.domain.create](#), [labkey.domain.createDesign](#), [labkey.domain.inferFields](#), [labkey.domain.save](#), [labkey.domain.drop](#), [labkey.domain.createConditionalFormatQueryFilter](#), [labkey.domain.FILTER\\_TYPES](#)

**Examples**

```
library(Rlabkey)

domain <- labkey.domain.get(baseUrl="http://labkey/", folderPath="home",
  schemaName="lists", queryName="test list")

## update the third field to use two conditional formats
qf <- labkey.domain.FILTER_TYPES
cf1 = labkey.domain.createConditionalFormat(labkey.domain.createConditionalFormatQueryFilter(qf$GT,
  100), bold=TRUE, text_color="D33115", background_color="333333")
cf2 = labkey.domain.createConditionalFormat(labkey.domain.createConditionalFormatQueryFilter(
  qf$LESS_THAN, 400), italic=TRUE, text_color="68BC00")
domain$fields$conditionalFormats[[3]] = rbind(cf1,cf2)

labkey.domain.save(baseUrl="http://labkey/", folderPath="home",
  schemaName="lists", queryName="test list", domainDesign=domain)
```

---

labkey.domain.createConditionalFormatQueryFilter  
*Create a conditional format query filter*

---

**Description**

Create a conditional format query filter string.

**Usage**

```
labkey.domain.createConditionalFormatQueryFilter(filterType, value,
  additionalFilter=NULL, additionalValue=NULL)
```

**Arguments**

filterType      a string specifying a permitted relational operator  
value            a string specifying a comparand  
additionalFilter      a string specifying a second relational operator  
additionalValue      a string specifying a second comparand

**Details**

This function can be used as a convenience wrapper to construct a query filter string for conditional formats. Two relational expressions may be formed, one with the first two parameters (for instance, parameter values '50' and 'eq' for value and filter respectively would create a condition of 'equals 50') and the second with the remaining two optional parameters. If both conditions are created, they are conjunct with a logical AND, and a value would have to pass both conditions to clear the filter. This function can be used in conjunction with `labkey.domain.FILTER_TYPES` for easy access to the set of permitted relational operators.

**Value**

The string specifying a query filter in LabKey filter URL format.

**Author(s)**

Rosaline Pyktel

**See Also**

[labkey.domain.get](#), [labkey.domain.create](#), [labkey.domain.createDesign](#), [labkey.domain.inferFields](#), [labkey.domain.save](#), [labkey.domain.drop](#), [labkey.domain.createConditionalFormat](#), [labkey.domain.FILTER\\_TYPES](#)

**Examples**

```
library(Rlabkey)

qf <- labkey.domain.FILTER_TYPES

# Filters for values equal to 750
qf1 <- labkey.domain.createConditionalFormatQueryFilter(qf$EQUAL, 750)
# Filters for values greater than 500, but less than 1000
qf2 <- labkey.domain.createConditionalFormatQueryFilter(qf$GREATER_THAN, 500, qf$LESS_THAN, 1000)
```

---

`labkey.domain.createDesign`*Helper function to create a domain design data structure*

---

### Description

Create a domain design data structure which can then be used by `labkey.domain.create` or `labkey.domain.save`

### Usage

```
labkey.domain.createDesign(name, description = NULL, fields, indices = NULL)
```

### Arguments

<code>name</code>	a string specifying the name of the domain
<code>description</code>	(optional) a string specifying domain description
<code>fields</code>	a list containing the fields of the domain design, this should be in the same format as returned by <code>labkey.inferFields</code> .
<code>indices</code>	(optional) a list of indices definitions to be used for this domain design on creation

### Details

This is a function which can be used to create a domain design data structure. Domain designs are used both when creating a new domain or updating an existing domain.

### Value

A list containing elements describing the domain design. Any of the APIs which take a domain design parameter can accept this data structure.

### Author(s)

Karl Lum

### See Also

[labkey.domain.get](#), [labkey.domain.inferFields](#), [labkey.domain.createIndices](#), [labkey.domain.create](#), [labkey.domain.save](#), [labkey.domain.drop](#), [labkey.domain.createConditionalFormat](#), [labkey.domain.createCon](#)  
[labkey.domain.FILTER\\_TYPES](#)

## Examples

```
## create a data frame and infer it's fields, then create a domain design from it
library(Rlabkey)

df <- data.frame(ptid=c(1:3), age = c(10,20,30), sex = c("f", "m", "f"))
fields <- labkey.domain.inferFields(baseUrl="http://labkey/", folderPath="home", df=df)
indices = labkey.domain.createIndices(list("ptid", "age"), TRUE)
indices = labkey.domain.createIndices(list("age"), FALSE, indices)
dd <- labkey.domain.createDesign(name="test list", fields=fields, indices=indices)
```

---

labkey.domain.createIndices

*Helper function to create a domain design indices list*

---

## Description

Create a list of indices definitions which can then be used by labkey.domain.createDesign

## Usage

```
labkey.domain.createIndices(colNames, asUnique, existingIndices = NULL)
```

## Arguments

colNames	a list of string column names for the index
asUnique	a logical TRUE or FALSE value for if a UNIQUE index should be used
existingIndices	a list of previously created indices definitions to append to

## Details

This helper function can be used to construct the list of indices definitions for a domain design structure. Each call to this function takes in the column names from the domain to use in the index and a parameter indicating if this should be a UNIQUE index. A third parameter can be used to build up more then one indices definitions.

## Value

The data frame containing the list of indices definitions, concatenated with the existingIndices object if provided.

## Author(s)

Cory Nathe

**See Also**

[labkey.domain.get](#), [labkey.domain.create](#), [labkey.domain.createDesign](#), [labkey.domain.inferFields](#), [labkey.domain.save](#), [labkey.domain.drop](#)

**Examples**

```
## create a list of indices definitions to use for a domain design
library(Rlabkey)

indices = labkey.domain.createIndices(list("intKey", "customInt"), TRUE)
indices = labkey.domain.createIndices(list("customInt"), FALSE, indices)
```

---

labkey.domain.drop      *Delete a LabKey domain*

---

**Description**

Delete an existing domain.

**Usage**

```
labkey.domain.drop(baseUrl=NULL, 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 name of the schema of the domain
queryName	a string specifying the query name

**Details**

This function will delete an existing domain along with any data that may have been uploaded to it.

**Author(s)**

Karl Lum

**See Also**

[labkey.domain.get](#), [labkey.domain.inferFields](#), [labkey.domain.createDesign](#), [labkey.domain.createIndices](#), [labkey.domain.save](#), [labkey.domain.create](#)



## Examples

```
## delete an existing domain
library(Rlabkey)

labkey.domain.drop(baseUrl="http://labkey/", folderPath="home",
  schemaName="lists", queryName="test list")
```

---

```
labkey.domain.FILTER_TYPES
```

*Provide comparator access*

---

## Description

A list specifying permitted validator comparators.

## Usage

```
labkey.domain.FILTER_TYPES
```

## Details

This constant contains a list specifying the set of permitted validator operators, using names to map conventional terms to the expressions used by LabKey filter URL formats. The values are intended to be used in conjunction with conditional formats or property validators.

## Value

A named list of strings.

## Author(s)

Rosaline Pyktel

## See Also

[labkey.domain.get](#), [labkey.domain.create](#), [labkey.domain.createDesign](#), [labkey.domain.inferFields](#), [labkey.domain.save](#), [labkey.domain.drop](#), [labkey.domain.createConditionalFormat](#), [labkey.domain.createCon](#)

## Examples

```
library(Rlabkey)

qf <- labkey.domain.FILTER_TYPES

# Example of available comparators
```

```
comparator1 <- qf$EQUAL
comparator2 <- qf$GREATER_THAN
comparator3 <- qf$DATE_LESS_THAN_OR_EQUAL
comparator4 <- qf$STARTS_WITH
comparator5 <- qf$CONTAINS_ONE_OF
```

---

labkey.domain.get      *Returns the metadata for an existing LabKey domain*

---

## Description

Get the data structure for a domain.

## Usage

```
labkey.domain.get(baseUrl=NULL, 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 name of the schema of the domain
queryName	a string specifying the query name

## Details

Returns the domain design of an existing domain. The returned domain design can be used for reporting purposes or it can be modified and used to create a new domain or update the domain source.

## Value

A list containing elements describing the domain. The structure is the same as a domain design created by `labkey.createDomainDesign`

## Author(s)

Karl Lum

## See Also

[labkey.domain.create](#), [labkey.domain.inferFields](#), [labkey.domain.createDesign](#), [labkey.domain.createIndices](#),  
[labkey.domain.save](#), [labkey.domain.drop](#)

## Examples

```
## retrieve an existing domain
library(Rlabkey)

labkey.domain.get(baseUrl="http://labkey/", folderPath="home",
  schemaName="lists", queryName="test list")
```

---

```
labkey.domain.inferFields
```

*Infer field metadata from a data frame*

---

## Description

Generate field information from the specified data frame. The resulting list can be used to create or edit a domain using the labkey.domain.create or labkey.domain.save APIs.

## Usage

```
labkey.domain.inferFields(baseUrl = NULL, folderPath, df)
```

## Arguments

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
df	a data frame specifying fields to infer. The data frame must have column names as well as row data to infer the type of the field from.

## Details

Field information can be generated from a data frame by introspecting the data associated with it along with other properties about that column. The data frame is posted to the server endpoint where the data is analyzed and returned as a list of fields each with its associated list of properties and values. This list can be edited and/or used to create a domain on the server.

## Value

The inferred metadata will be returned as a list with an element called : "fields" which contains the list of fields inferred from the data frame. Each field will contain the list of attributes and values for that field definition.

## Author(s)

Karl Lum

**See Also**

[labkey.domain.get](#), [labkey.domain.create](#), [labkey.domain.createDesign](#), [labkey.domain.createIndices](#), [labkey.domain.save](#), [labkey.domain.drop](#)

**Examples**

```
## create a data frame and infer it's fields
library(Rlabkey)

df <- data.frame(ptid=c(1:3), age = c(10,20,30), sex = c("f", "m", "f"))
fields <- labkey.domain.inferFields(baseUrl="http://labkey/", folderPath="home", df=df)
```

---

labkey.domain.save      *Updates an existing LabKey domain*

---

**Description**

Modify an existing domain with the specified domain design.

**Usage**

```
labkey.domain.save(baseUrl=NULL, folderPath, schemaName, queryName, domainDesign)
```

**Arguments**

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
schemaName	a string specifying the name of the schema of the domain
queryName	a string specifying the query name
domainDesign	a list data structure with the domain design to update to

**Value**

A list containing elements describing the domain after the update. The structure is the same as a domain design created by `labkey.createDomainDesign`

**Author(s)**

Karl Lum

**See Also**

[labkey.domain.get](#), [labkey.domain.inferFields](#), [labkey.domain.createDesign](#), [labkey.domain.createIndices](#), [labkey.domain.create](#), [labkey.domain.drop](#), [labkey.domain.createConditionalFormat](#), [labkey.domain.createC](#), [labkey.domain.FILTER\\_TYPES](#)

**Examples**

```
library(Rlabkey)
## change the type of one of the columns
domain <- labkey.domain.get(baseUrl="http://labkey/", folderPath="home",
  schemaName="lists", queryName="test list")

domain$fields[3,]$rangeURI = "xsd:string"
domain$fields[3,]$name = "changed to string"

labkey.domain.save(baseUrl="http://labkey/", folderPath="home",
  schemaName="lists", queryName="test list", domainDesign=domain)
```

---

<code>labkey.executeSql</code>	<i>Retrieve data from a LabKey Server using SQL commands</i>
--------------------------------	--

---

**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, colSort=NULL,
  showHidden = FALSE, colNameOpt='caption',
  containerFilter=NULL, parameters=NULL)
```

**Arguments**

<code>baseUrl</code>	a string specifying the baseUrl for the labkey server
<code>folderPath</code>	a string specifying the folderPath
<code>schemaName</code>	a string specifying the schemaName for the query
<code>sql</code>	a string containing the sql commands to be executed
<code>maxRows</code>	(optional) an integer specifying the maximum number of rows to return. If no value is specified, all rows are returned.
<code>rowOffset</code>	(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.

colSort	(optional) a string including the name of the column to sort preceded by a “+” or “-” to indicate sort direction
showHidden	(optional) a logical value indicating whether or not to return data columns that would normally be hidden from user view. Defaults to FALSE if no value provided.
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.
containerFilter	(optional) Specifies the containers to include in the scope of selectRows request. A value of NULL is equivalent to "Current". Valid values are <ul style="list-style-type: none"> <li>• "Current": Include the current folder only</li> <li>• "CurrentAndSubfolders": Include the current folder and all subfolders</li> <li>• "CurrentPlusProject": Include the current folder and the project that contains it</li> <li>• "CurrentAndParents": Include the current folder and its parent folders</li> <li>• "CurrentPlusProjectAndShared": Include the current folder plus its project plus any shared folders</li> <li>• "AllFolders": Include all folders for which the user has read permission</li> </ul>
parameters	(optional) List of name/value pairs for the parameters if the SQL references underlying queries that are parameterized. For example, parameters=c("X=1", "Y=2").

### 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.importRows](#), [labkey.updateRows](#), [labkey.deleteRows](#), [getRows](#)

## Examples

```
## 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
```

---

```
labkey.experiment.createData
```

*Create an experiment data object*

---

## Description

Create an experiment data object.

## Usage

```
labkey.experiment.createData(config,
  dataClassId = NULL, dataClassName = NULL, dataFileUrl = NULL)
```

## Arguments

config	a list of base experiment object properties
dataClassId	(optional) a integer specifying the data class row ID
dataClassName	(optional) a string specifying the name of the data class
dataFileUrl	(optional) a string specifying the local file url of the uploaded file

## Details

Create an experiment data object which can be used as either input or output datas for an experiment run.

**Value**

Returns the object representation of the experiment data object.

**Author(s)**

Karl Lum

**See Also**

[labkey.experiment.saveBatch](#), [labkey.experiment.createMaterial](#), [labkey.experiment.createRun](#)

**Examples**

```
library(Rlabkey)

## create a non-assay backed run with data classes as data inputs and outputs

d1 <- labkey.experiment.createData(
  list(name = "dc-01"), dataClassId = 400)
d2 <- labkey.experiment.createData(
  list(name = "dc-02"), dataClassId = 402)
run <- labkey.experiment.createRun(
  list(name="new run"), dataInputs = d1, dataOutputs = d2)
labkey.experiment.saveBatch(baseUrl="http://labkey/", folderPath="home",
  protocolName=labkey.experiment.SAMPLE_DERIVATION_PROTOCOL, runList=run)
```

---

```
labkey.experiment.createMaterial
```

*Create an experiment material object*

---

**Description**

Create an experiment material object.

**Usage**

```
labkey.experiment.createMaterial(config, sampleSetId = NULL, sampleSetName = NULL)
```

**Arguments**

config	a list of base experiment object properties
sampleSetId	(optional) a integer specifying the sample set row ID
sampleSetName	(optional) a string specifying the name of the sample set



**Details**

Create an experiment material object which can be used as either input or output materials for an experiment run.

**Value**

Returns the object representation of the experiment material object.

**Author(s)**

Karl Lum

**See Also**

[labkey.experiment.saveBatch](#), [labkey.experiment.createData](#), [labkey.experiment.createRun](#)

**Examples**

```
library(Rlabkey)

## create a non-assay backed run with samples as material inputs and outputs

m1 <- labkey.experiment.createMaterial(
  list(name = "87444063.2604.626"), sampleSetName = "Study Specimens")
m2 <- labkey.experiment.createMaterial(
  list(name = "87444063.2604.625"), sampleSetName = "Study Specimens")
run <- labkey.experiment.createRun(
  list(name="new run"), materialInputs = m1, materialOutputs = m2)
labkey.experiment.saveBatch(baseUrl="http://labkey/", folderPath="home",
  protocolName=labkey.experiment.SAMPLE_DERIVATION_PROTOCOL, runList=run)
```

---

labkey.experiment.createRun

*Create an experiment run object*

---

**Description**

Create an experiment run object.

**Usage**

```
labkey.experiment.createRun(config,
  dataInputs = NULL, dataOutputs = NULL, dataRows = NULL,
  materialInputs = NULL, materialOutputs = NULL, plateMetadata = NULL)
```

**Arguments**

<code>config</code>	a list of base experiment object properties
<code>dataInputs</code>	(optional) a list of experiment data objects to be used as data inputs to the run
<code>dataOutputs</code>	(optional) a list of experiment data objects to be used as data outputs to the run
<code>dataRows</code>	(optional) a data frame containing data rows to be uploaded to the assay backed run
<code>materialInputs</code>	(optional) a list of experiment material objects to be used as material inputs to the run
<code>materialOutputs</code>	(optional) a list of experiment material objects to be used as material outputs to the run
<code>plateMetadata</code>	(optional) if the assay supports plate templates, the plate metadata object to upload

**Details**

Create an experiment run object which can be used in the `saveBatch` function.

**Value**

Returns the object representation of the experiment run object.

**Author(s)**

Karl Lum

**See Also**

[labkey.experiment.saveBatch](#), [labkey.experiment.createData](#), [labkey.experiment.createMaterial](#)

**Examples**

```
library(Rlabkey)

## create a non-assay backed run with samples as material inputs and outputs

m1 <- labkey.experiment.createMaterial(
  list(name = "87444063.2604.626"), sampleSetName = "Study Specimens")
m2 <- labkey.experiment.createMaterial(
  list(name = "87444063.2604.625"), sampleSetName = "Study Specimens")
run <- labkey.experiment.createRun(
  list(name="new run"), materialInputs = m1, materialOutputs = m2)
labkey.experiment.saveBatch(baseUrl="http://labkey/", folderPath="home",
  protocolName=labkey.experiment.SAMPLE_DERIVATION_PROTOCOL, runList=run)

## import an assay run which includes plate metadata
```

```

df <- data.frame(participantId=c(1:3), visitId = c(10,20,30), welllocation = c("A1", "D11", "F12"))

runConfig <- fromJSON(txt='{
  "assayId": 310,
  "name" : "api imported run with plate metadata",
  "properties" : {
    "PlateTemplate" : "urn:lsid:labkey.com:PlateTemplate.Folder-6:d8bbec7d-34cd-1038-bd67-b3bd"
  }
}')

plateMetadata <- fromJSON(txt='{
  "control" : {
    "positive" : {"dilution": 0.005},
    "negative" : {"dilution": 1.0}
  },
  "sample" : {
    "SA01" : {"dilution": 1.0, "ID" : 111, "Barcode" : "BC_111", "Concentration" : 0.0125},
    "SA02" : {"dilution": 2.0, "ID" : 222, "Barcode" : "BC_222"},
    "SA03" : {"dilution": 3.0, "ID" : 333, "Barcode" : "BC_333"},
    "SA04" : {"dilution": 4.0, "ID" : 444, "Barcode" : "BC_444"}
  }
}')

run <- labkey.experiment.createRun(runConfig, dataRows = df, plateMetadata = plateMetadata)
labkey.experiment.saveBatch(
  baseUrl="http://labkey/", folderPath="home",
  assayConfig=list(assayId = 310), runList=run
)

```

---

labkey.experiment.SAMPLE\_DERIVATION\_PROTOCOL

*Constant for the Simple Derivation Protocol*

---

## Description

Simple Derivation Protocol constant.

## Details

This value can be used in the `labkey.experiment.saveBatch` function when creating runs that aren't backed by an assay protocol.

## Author(s)

Karl Lum

## See Also

[labkey.experiment.saveBatch](#)

---

```
labkey.experiment.saveBatch
```

*Saves a modified experiment batch*

---

### Description

Saves a modified experiment batch.

### Usage

```
labkey.experiment.saveBatch(baseUrl=NULL, folderPath,  
    assayConfig = NULL, protocolName = NULL,  
    batchPropertyList = NULL, runList)
```

### Arguments

<code>baseUrl</code>	(optional) a string specifying the baseUrl for the labkey server
<code>folderPath</code>	a string specifying the folderPath
<code>assayConfig</code>	(optional) a list specifying assay configuration information
<code>protocolName</code>	(optional) a string specifying the protocol name of the protocol to use
<code>batchPropertyList</code>	(optional) a list of batch properties
<code>runList</code>	a list of experiment run objects

### Details

Saves a modified batch. Runs within the batch may refer to existing data and material objects, either inputs or outputs, by ID or LSID. Runs may also define new data and materials objects by not specifying an ID or LSID in their properties.

Runs can be created for either assay or non-assay backed protocols. For an assay backed protocol, either the `assayId` or the `assayName` and `providerName` name must be specified in the `assayConfig` parameter. If a non-assay backed protocol is to be used, specify the `protocolName` string value, note that currently only the simple : `labkey.experiment.SAMPLE_DERIVATION_PROTOCOL` is supported.

Refer to the `labkey.experiment.createData`, `labkey.experiment.createMaterial`, and `labkey.experiment.createRun` helper functions to assemble the data structure that `saveBatch` expects.

### Value

Returns the object representation of the experiment batch.

### Author(s)

Karl Lum

**See Also**

[labkey.experiment.createData](#), [labkey.experiment.createMaterial](#), [labkey.experiment.createRun](#)

**Examples**

```
library(Rlabkey)

## uploads data to an existing assay

df <- data.frame(participantId=c(1:3), visitId = c(10,20,30), sex = c("f", "m", "f"))
bprops <- list(LabNotes="this is a simple demo")
bpl <- list(name=paste("Batch ", as.character(date())),properties=bprops)
run <- labkey.experiment.createRun(list(name="new assay run"), dataRows = df)
labkey.experiment.saveBatch(baseUrl="http://labkey/", folderPath="home",
  assayConfig=list(assayName="GPAT", providerName="General"),
  batchPropertyList=bpl, runList=run)

## create a non-assay backed run with samples as material inputs and outputs

m1 <- labkey.experiment.createMaterial(
  list(name = "87444063.2604.626"), sampleSetName = "Study Specimens")
m2 <- labkey.experiment.createMaterial(
  list(name = "87444063.2604.625"), sampleSetName = "Study Specimens")
run <- labkey.experiment.createRun(
  list(name="new run"), materialInputs = m1, materialOutputs = m2)
labkey.experiment.saveBatch(baseUrl="http://labkey/", folderPath="home",
  protocolName=labkey.experiment.SAMPLE_DERIVATION_PROTOCOL, runList=run)

## import an assay run which includes plate metadata

df <- data.frame(participantId=c(1:3), visitId = c(10,20,30), welllocation = c("A1", "D11", "F12"))

runConfig <- fromJSON(txt='{"assayId": 310,
  "name" : "api imported run with plate metadata",
  "properties" : {
    "PlateTemplate" : "urn:lsid:labkey.com:PlateTemplate.Folder-6:d8bbec7d-34cd-1038-bd67-b3bd"
  }}')

plateMetadata <- fromJSON(txt='{
  "control" : {
    "positive" : {"dilution": 0.005},
    "negative" : {"dilution": 1.0}
  },
  "sample" : {
    "SA01" : {"dilution": 1.0, "ID" : 111, "Barcode" : "BC_111", "Concentration" : 0.0125},
    "SA02" : {"dilution": 2.0, "ID" : 222, "Barcode" : "BC_222"},
    "SA03" : {"dilution": 3.0, "ID" : 333, "Barcode" : "BC_333"},
    "SA04" : {"dilution": 4.0, "ID" : 444, "Barcode" : "BC_444"}
  }
}
```

```
    }')  
  
    run <- labkey.experiment.createRun(runConfig, dataRows = df, plateMetadata = plateMetadata)  
    labkey.experiment.saveBatch(  
      baseUrl="http://labkey/", folderPath="home",  
      assayConfig=list(assayId = 310), runList=run  
    )
```

---

labkey.experiment.saveRuns  
*Saves Runs.*

---

### Description

Saves experiment runs.

### Usage

```
labkey.experiment.saveRuns(baseUrl=NULL, folderPath,  
  protocolName, runList)
```

### Arguments

baseUrl	(optional) a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
protocolName	a string specifying the protocol name of the protocol to use
runList	a list of experiment run objects

### Details

Saves experiment runs. Runs may refer to existing data and material objects, either inputs or outputs, by ID or LSID. Runs may also define new data and materials objects by not specifying an ID or LSID in their properties.

Refer to the labkey.experiment.createData, labkey.experiment.createMaterial, and labkey.experiment.createRun helper functions to assemble the data structure that saveRuns expects.

### Value

Returns the object representation of the experiment run.

### Author(s)

Ankur Juneja

**See Also**

[labkey.experiment.createData](#), [labkey.experiment.createMaterial](#), [labkey.experiment.createRun](#)

**Examples**

```
library(Rlabkey)

## example with materialInputs and materialOutputs

m1 <- labkey.experiment.createMaterial(
  list(name = "87444063.2604.626"), sampleSetName = "Study Specimens")
m2 <- labkey.experiment.createMaterial(
  list(name = "87444063.2604.625"), sampleSetName = "Study Specimens")
run <- labkey.experiment.createRun(
  list(name="new run"), materialInputs = m1, materialOutputs = m2)
labkey.experiment.saveRuns(baseUrl="http://labkey/", folderPath="home",
  protocolName=labkey.experiment.SAMPLE_DERIVATION_PROTOCOL, runList=run)
```

---

labkey.getBaseUrl	<i>Get the default baseUrl parameter used for all http or https requests</i>
-------------------	--

---

**Description**

Use this function to get "baseUrl" package environment variables to be used for all http or https requests.

**Usage**

```
labkey.getBaseUrl(baseUrl=NULL)
```

**Arguments**

baseUrl            server location including context path, if any. e.g. <https://www.labkey.org/>

**Details**

The function takes an optional baseUrl parameter. When non empty parameter is passed in and if baseUrl has not been previously set, the function will remember the baseUrl value in package environment variables and return the formatted baseUrl. Skip baseUrl parameter to get previously set baseUrl.

## Examples

```
## Example of getting previously set baseUrl
library(Rlabkey)
labkey.setDefaults(apiKey="session|abcdef0123456789abcdef0123456789",
  baseUrl="http://labkey/")
labkey.getBaseUrl()
```

---

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 `labkey.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.importRows](#), [labkey.deleteRows](#)

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

**Examples**

```
## 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
```

---

labkey.getFolders	<i>Retrieve a list of folders accessible to the current user</i>
-------------------	--

---

**Description**

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

**Usage**

```
labkey.getFolders(baseUrl, folderPath,
  includeEffectivePermissions=TRUE,
  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.
includeEffectivePermissions	If set to false, the effective permissions for this container resource will not be included. (defaults to TRUE).
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 folders are returned as a three-column data frame containing

name	the name of the folder
folderPath	the full path of the folder from the project root
effectivePermissions	the current user's effective permissions for the given folder

## Author(s)

Peter Hussey, peter@labkey.com

## See Also

[labkey.getQueries](#), [labkey.getQueryViews](#), [labkey.getQueryDetails](#), [labkey.getDefaultViewDetails](#), [labkey.getLookupDetails](#), [labkey.security.getContainers](#), [labkey.security.createContainer](#), [labkey.security.deleteContainer](#), [labkey.security.moveContainer](#)

## Examples

```
## List of folders
# library(Rlabkey)

folders <- labkey.getFolders("https://www.labkey.org", "/home")
folders
```

---

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**

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 the query
lookupKey	a string specifying the qualified name of a lookup field (foreign key) relative to the query specified by queryName

**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.importRows](#), [labkey.deleteRows](#)

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

**Examples**

```
## 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 the 2nd
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

```

---

```
labkey.getModuleProperty
```

*Get effective module property value*

---

### Description

Get a specific effective module property value for folder

### Usage

```
labkey.getModuleProperty(baseUrl=NULL, folderPath, moduleName, propName)
```

### Arguments

baseUrl	server location including context path, if any. e.g. <a href="https://www.labkey.org/">https://www.labkey.org/</a>
folderPath	a string specifying the folderPath
moduleName	name of the module
propName	The module property name

**Examples**

```
library(Rlabkey)
labkey.getModuleProperty(baseUrl="http://labkey/", folderPath="flowProject",
  moduleName="flow", propName="ExportToScriptFormat")
```

---

labkey.getQueries	<i>Retrieve a list of available queries for a specified LabKey schema</i>
-------------------	---

---

**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.net/RCurl/>,  
<https://www.labkey.org/project/home/begin.view>

## See Also

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

## Examples

```
## List of queries in a schema
# library(Rlabkey)

queriesDF <- labkey.getQueries(
  baseUrl="https://www.labkey.org",
  folderPath="/home",
  schemaName="lists"
)
queriesDF
```

---

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 issued 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.importRows](#), [labkey.deleteRows](#)

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

### Examples

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

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

---

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**

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 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/Documentation/wiki-page.view?name=savingViews>

**See Also**

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

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

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

**Examples**

```
## 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"  
)
```

---

labkey.getRequestOptions

*Helper function to get the HTTP request options for a specific method type.*

---

### Description

The internal functions for labkey.get() and labkey.post() use this labkey.getRequestOptions() helper to build up the HTTP request options for things like CSRF, CURL options, and authentication properties. This function is also exposed for general use if you would like to make your own HTTP request but need to use those request options as set in your session context.

### Usage

```
labkey.getRequestOptions(method = 'GET', encoding = NULL)
```

### Arguments

method	a string specifying the HTTP method for the request options you want to get
encoding	a string specifying the type of encoding to add to the header properties, defaults to UTF-8 when NULL

### Author(s)

Cory Nathe

### Examples

```
library(Rlabkey)  
labkey.getRequestOptions()
```

---

labkey.getSchemas	<i>Retrieve a list of available schemas from a labkey database</i>
-------------------	--

---

### 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 generatly associated with a LabKey Server "module" that provides some specific functionality. Within a querable 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.net/RCurl/>,  
<https://www.labkey.org/project/home/begin.view>

### See Also

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

## Examples

```
## List of schemas
# library(Rlabkey)

schemasDF <- labkey.getSchemas(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples"
)
```

---

labkey.importRows      *Import rows of data into a LabKey Server*

---

## Description

Bulk import rows of data into the database.

## Usage

```
labkey.importRows(baseUrl, folderPath,
  schemaName, queryName, toImport, na)
```

## 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
toImport	a data frame containing rows of data to be imported
na	(optional) the value to convert NA's to, defaults to NULL

## Details

Multiple rows of data can be imported in bulk. The toImport 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 import a value of NULL, use an empty string ("") in the data frame (regardless of the database column type). Also, when importing data into a study dataset, the sequence number must be specified.

## Value

A list is returned with named categories of **command**, **rowsAffected**, **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.

**Author(s)**

Cory Nathe

**See Also**

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

**Examples**

```
## 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)

newrows <- data.frame(
  DisplayFld="Imported from R"
  , RequiredText="abc"
  , RequiredInt=1
  , stringsAsFactors=FALSE)
newrows = newrows[rep(1:nrow(newrows),each=5),]

importedInfo <- labkey.importRows("http://localhost:8080/labkey",
  folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
  toImport=newrows)

importedInfo$rowsAffected
```

---

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, na, provenanceParams=NULL)
```

**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
na	(optional) the value to convert NA's to, defaults to NULL
provenanceParams	the provenance parameter object which contains the options to include as part of a provenance recording. This is a premium feature and requires the Provenance LabKey module to function correctly, if it is not present this parameter will be ignored.

### 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.importRows](#), [labkey.updateRows](#), [labkey.deleteRows](#), [labkey.query.import](#), [labkey.provenance.createProvenanceParams](#), [labkey.provenance.startRecording](#), [labkey.provenance.addRecordingStep](#), [labkey.provenance.stopRecording](#)

### Examples

```
## 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", queryName="AllTypes",
  toInsert=newrow)
newRowId <- insertedRow$rows[[1]]$RowId

selectedRow<-labkey.selectRows("http://localhost:8080/labkey",
  folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
  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", queryName="AllTypes",
  toUpdate=updaterow)
selectedRow<-labkey.selectRows("http://localhost:8080/labkey",
  folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
  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", queryName="AllTypes",
  toDelete=deleterow)

## Example of creating a provenance run with an initial step with material inputs, a second step
## with provenance mapping to link existing samples with newly inserted samples, and a final step
## with a data output
##
mi <- data.frame(lsid=c("urn:lsid:labkey.com:Sample.251.MySamples:sample1",
  "urn:lsid:labkey.com:Sample.251.MySamples:sample2"))
p <- labkey.provenance.createProvenanceParams(name="step1", description="initial step",
  materialInputs=mi)
ra <- labkey.provenance.startRecording(baseUrl="https://labkey.org/labkey/",
  folderPath = "Provenance", provenanceParams=p)

rows <- fromJSON(txt='[{
  "name" : "sample3",
  "protein" : "p3",
  "prov:objectInputs" : [
    "urn:lsid:labkey.com:Sample.251.MySamples:sample21",
    "urn:lsid:labkey.com:Sample.251.MySamples:sample22"
  ]
}]')

```

```

    ]
  },{
    "name" : "sample4",
    "protein" : "p4",
    "prov:objectInputs" : [
      "urn:lsid:labkey.com:Sample.251.MySamples:sample21",
      "urn:lsid:labkey.com:Sample.251.MySamples:sample22"
    ]
  }
]')
labkey.insertRows(baseUrl="https://labkey.org/labkey/", folderPath = "Provenance",
  schemaName="samples", queryName="MySamples", toInsert=rows,
  provenanceParams=labkey.provenance.createProvenanceParams(name="query step",
    recordingId=ra$recordingId))
labkey.provenance.stopRecording(baseUrl="https://labkey.org/labkey/", folderPath = "Provenance",
  provenanceParams=labkey.provenance.createProvenanceParams(name="final step",
    recordingId=ra$recordingId, dataOutputs=do))

```

---

labkey.makeRemotePath *Build a file path to data on a remote machine*

---

### Description

Replaces a local root with a remote root given a full path

### Usage

```
labkey.makeRemotePath(localRoot, remoteRoot, fullPath)
```

### Arguments

localRoot	local root part of the fullPath
remoteRoot	remote root that will replace the local root of the fullPath
fullPath	the full path to make remote

### Details

A helper function to translate a file path on a LabKey web server to a path accessible by a remote machine. For example, if an R script is run on an R server that is a different machine than the LabKey server and that script references data files on the LabKey server, a remote path needs to be created to correctly reference these files. The local and remote roots of the data pipeline are included by LabKey in the prolog of an R View report script. Note that the data pipeline root references are only included if an administrator has enabled the Rserve Reports experimental feature on the LabKey server. If the remoteRoot is empty or the fullPath does not contain the localRoot then the fullPath is returned without its root being changed.



**Value**

A character array containing the full path.

**Author(s)**

Dax Hawkins

**Examples**

```
# library(Rlabkey)

labkey.pipeline.root <- "c:/data/fcs"
labkey.remote.pipeline.root <- "/volumes/fcs"
fcsFile <- "c:/data/fcs/runA/aaa.fcs"

# returns "/volumes/fcs/runA/aaa.fcs"
labkey.makeRemotePath(
  localRoot=labkey.pipeline.root,
  remoteRoot=labkey.remote.pipeline.root,
  fullPath=fcsFile);
```

---

labkey.pipeline.getFileStatus

*Gets the protocol file status for a pipeline*

---

**Description**

Gets the status of analysis using a particular protocol for a particular pipeline.

**Usage**

```
labkey.pipeline.getFileStatus(baseUrl=NULL, folderPath,
  taskId, protocolName, path, files)
```

**Arguments**

baseUrl	a string specifying the baseUrl for the LabKey server
folderPath	a string specifying the folderPath
taskId	a string identifier for the pipeline
protocolName	a string name of the analysis protocol
path	a string for the relative path from the folder's pipeline root
files	a list of names of the files within the subdirectory described by the path property

**Value**

The response will contain a list of file status objects, i.e. files, each of which will have the following properties:

- "name": name of the file
- "status": status of the file

The response will also include the name of the action that would be performed on the files if the user initiated processing, i.e. submitType.

**Author(s)**

Cory Nathe

**See Also**

[labkey.pipeline.getPipelineContainer](#), [labkey.pipeline.getProtocols](#), [labkey.pipeline.startAnalysis](#)

**Examples**

```
labkey.pipeline.getFileStatus(  
    baseUrl="http://labkey/",  
    folderPath="home",  
    taskId = "pipelinetest:pipeline:r-copy",  
    path = "r-copy",  
    protocolName = "Test protocol name",  
    files = list("sample.txt", "result.txt")  
)
```

---

```
labkey.pipeline.getPipelineContainer
```

*Gets the container in which the pipeline is defined*

---

**Description**

Gets the container in which the pipeline for this container is defined. This may be the container in which the request was made, or a parent container if the pipeline was defined there.

**Usage**

```
labkey.pipeline.getPipelineContainer(baseUrl=NULL, folderPath)
```

### Arguments

baseUrl            a string specifying the baseUrl for the LabKey server  
folderPath        a string specifying the folderPath

### Value

The response will contain the following:

- "containerPath": The container path in which the pipeline is defined. If no pipeline has been defined in this container hierarchy, then the value of this property will be null.
- "webDavURL": The WebDavURL for the pipeline root.

### Author(s)

Cory Nathe

### See Also

[labkey.pipeline.getProtocols](#), [labkey.pipeline.getFileStatus](#), [labkey.pipeline.startAnalysis](#)

### Examples

```
labkey.pipeline.getPipelineContainer(  
    baseUrl="http://labkey/",  
    folderPath="home"  
)
```

---

labkey.pipeline.getProtocols

*Gets the protocols that have been saved for a particular pipeline*

---

### Description

Gets the protocols that have been saved for a particular pipeline.

### Usage

```
labkey.pipeline.getProtocols(baseUrl=NULL, folderPath,  
    taskId, path, includeWorkbooks = FALSE)
```

**Arguments**

<code>baseUrl</code>	a string specifying the baseUrl for the LabKey server
<code>folderPath</code>	a string specifying the folderPath
<code>taskId</code>	a string identifier for the pipeline
<code>path</code>	a string for the relative path from the folder's pipeline root
<code>includeWorkbooks</code>	(optional) If true, protocols from workbooks under the selected container will also be included. Defaults to FALSE.

**Value**

The response will contain a list of protocol objects, each of which will have the following properties:

- "name": Name of the saved protocol.
- "description": Description of the saved protocol, if provided.
- "xmlParameters": Bioml representation of the parameters defined by this protocol.
- "jsonParameters": A list representation of the parameters defined by this protocol.
- "containerPath": The container path where this protocol was saved.

The response will also include a `defaultProtocolName` property representing which of the protocol names is the default.

**Author(s)**

Cory Nathe

**See Also**

[labkey.pipeline.getPipelineContainer](#), [labkey.pipeline.getFileStatus](#), [labkey.pipeline.startAnalysis](#)

**Examples**

```
labkey.pipeline.getProtocols(  
  baseUrl="http://labkey/",  
  folderPath="home",  
  taskId = "pipelinetest:pipeline:r-copy",  
  path = "r-copy",  
  includeWorkbooks = FALSE  
)
```

---

 labkey.pipeline.startAnalysis

*Start an analysis of a set of files using a pipeline*


---

### Description

Starts analysis of a set of files using a particular protocol definition with a particular pipeline.

### Usage

```
labkey.pipeline.startAnalysis(baseUrl=NULL, folderPath,
  taskId, protocolName, path, files, fileIds = list(),
  pipelineDescription = NULL, protocolDescription = NULL,
  jsonParameters = NULL, xmlParameters = NULL,
  allowNonExistentFiles = FALSE, saveProtocol = TRUE)
```

### Arguments

baseUrl	a string specifying the baseUrl for the LabKey server
folderPath	a string specifying the folderPath
taskId	a string identifier for the pipeline
protocolName	a string name of the analysis protocol
path	a string for the relative path from the folder's pipeline root
files	a list of names of the files within the subdirectory described by the path property
fileIds	(optional) list of data IDs of files to be used as inputs for this pipeline. These correspond to the rowIds from the table ext.data. They do not need to be located within the file path provided. The user does need read access to the container associated with each file.
pipelineDescription	(optional) a string description displayed in the pipeline
protocolDescription	(optional) a string description of the analysis protocol
jsonParameters	(optional) a list of key / value pairs, or a JSON string representation, for the protocol description. Not allowed if a protocol with the same name has already been saved. If no protocol with the same name exists, either this property or xmlParameters must be specified.
xmlParameters	(optional) a string XML representation of the protocol description. Not allowed if a protocol with the same name has already been saved. If no protocol with the same name exists, either this property or jsonParameters must be specified.
allowNonExistentFiles	(optional) a boolean indicating if the pipeline should allow non existent files. Defaults to false.
saveProtocol	(optional) a boolean indicating if no protocol with this name already exists, whether or not to save this protocol definition for future use. Defaults to true.

**Value**

On success, the response will contain the jobGUID string value for the newly created pipeline job.

**Author(s)**

Cory Nathe

**See Also**

[labkey.pipeline.getPipelineContainer](#), [labkey.pipeline.getProtocols](#), [labkey.pipeline.getFileStatus](#)

**Examples**

```
labkey.pipeline.startAnalysis(  
    baseUrl="http://labkey/",  
    folderPath="home",  
    taskId = "pipelinetest:pipeline:r-copy",  
    protocolName = "Test protocol name",  
    path="r-copy",  
    files = list("sample.txt", "result.txt"),  
    protocolDescription = "Test protocol description",  
    pipelineDescription = "test pipeline description",  
    jsonParameters = list(assay = "Test assay name", comment = "Test assay comment"),  
    saveProtocol = TRUE  
)
```

---

```
labkey.provenance.addRecordingStep
```

*Add a step to a provenance recording*

---

**Description**

Function to add a step to a previously created provenance recording session. Note: this function is in beta and not yet final, changes should be expected so exercise caution when using it.

**Usage**

```
labkey.provenance.addRecordingStep(baseUrl=NULL, folderPath, provenanceParams = NULL)
```

**Arguments**

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
provenanceParams	the provenance parameter object which contains the options to include in this recording step

## Details

Function to add a step to a previously created provenance recording. The recording ID that was obtained from a previous startRecording function call must be passed into the provenanceParams config. This is a premium feature and requires the Provenance LabKey module to function correctly.

## Value

The generated recording ID which can be used in subsequent steps (or queries that support provenance).

## Author(s)

Karl Lum

## See Also

[labkey.provenance.createProvenanceParams](#), [labkey.provenance.startRecording](#), [labkey.provenance.stopRecording](#)

## Examples

```
## start a provenance recording and add a recording step
library(Rlabkey)

mi <- data.frame(lsid=c("urn:lsid:labkey.com:Sample.251.MySamples:sample1",
  "urn:lsid:labkey.com:Sample.251.MySamples:sample2"))

p <- labkey.provenance.createProvenanceParams(name="step1", description="initial step",
  materialInputs=mi)
r <- labkey.provenance.startRecording(baseUrl="https://labkey.org/labkey/",
  folderPath = "Provenance", provenanceParams=p)
do <- data.frame(
  lsid="urn:lsid:labkey.com:AssayRunTSVData.Folder-251:12c70994-7ce5-1038-82f0-9c1487dbd334")

labkey.provenance.addRecordingStep(baseUrl="https://labkey.org/labkey/", folderPath = "Provenance",
  provenanceParams=labkey.provenance.createProvenanceParams(name="additional step",
  recordingId=r$recordingId, dataOutputs=do))
```

---

labkey.provenance.createProvenanceParams

*Create provenance parameter object*

---

## Description

Helper function to create the data structure that can be used in provenance related APIs. Note: this function is in beta and not yet final, changes should be expected so exercise caution when using it.

**Usage**

```
labkey.provenance.createProvenanceParams(recordingId=NULL, name=NULL, description=NULL,
materialInputs=NULL, materialOutputs=NULL, dataInputs=NULL, dataOutputs=NULL,
inputObjectUriProperty=NULL, outputObjectUriProperty=NULL, objectInputs=NULL,
objectOutputs=NULL, provenanceMap=NULL)
```

**Arguments**

<code>recordingId</code>	(optional) the recording ID to associate with other steps using the same ID
<code>name</code>	(optional) the name of this provenance step
<code>description</code>	(optional) the description of this provenance step
<code>materialInputs</code>	(optional) the list of materials (samples) to be used as the provenance run input. The data structure should be a dataframe with the column name describing the data type (lsid, id)
<code>materialOutputs</code>	(optional) the list of materials (samples) to be used as the provenance run output. The data structure should be a dataframe with the column name describing the data type (lsid, id)
<code>dataInputs</code>	(optional) the list of data inputs to be used for the run provenance map
<code>dataOutputs</code>	(optional) the list of data outputs to be used for the run provenance map
<code>inputObjectUriProperty</code>	(optional) for incoming data rows, the column name to interpret as the input to the provenance map. Defaults to : 'prov:objectInputs'
<code>outputObjectUriProperty</code>	(optional) for provenance mapping, the column name to interpret as the output to the provenance map. Defaults to : 'lsid'
<code>objectInputs</code>	(optional) the list of object inputs to be used for the run provenance map
<code>objectOutputs</code>	(optional) the list of object outputs to be used for the run provenance map
<code>provenanceMap</code>	(optional) the provenance map to be used directly for the run step. The data structure should be a dataframe with the column names of 'from' and 'to' to indicate which sides of the mapping the identifiers refer to

**Details**

This function can be used to generate a provenance parameter object which can then be used as an argument in the other provenance related functions to assemble provenance runs. This is a premium feature and requires the Provenance LabKey module to function correctly.

**Value**

A list containing elements describing the passed in provenance parameters.

**Author(s)**

Karl Lum



**See Also**

[labkey.provenance.startRecording](#), [labkey.provenance.addRecordingStep](#), [labkey.provenance.stopRecording](#)

**Examples**

```
## create provenance params with material inputs and data outputs
library(Rlabkey)

mi <- data.frame(lsid=c("urn:lsid:labkey.com:Sample.251.MySamples:sample1",
  "urn:lsid:labkey.com:Sample.251.MySamples:sample2"))
do <- data.frame(
  lsid="urn:lsid:labkey.com:AssayRunTSVData.Folder-251:12c70994-7ce5-1038-82f0-9c1487dbd334")

p <- labkey.provenance.createProvenanceParams(name="step1", description="initial step",
  materialInputs=mi, dataOutputs=do)

## create provenance params with object inputs (from an assay run)
oi <- labkey.selectRows(baseUrl="https://labkey.org/labkey/", folderPath = "Provenance",
  schemaName="assay.General.titer",
  queryName="Data",
  colSelect= c("LSID"),
  colFilter=makeFilter(c("Run/RowId","EQUAL","253")))
mi <- data.frame(lsid=c("urn:lsid:labkey.com:Sample.251.MySamples:sample1",
  "urn:lsid:labkey.com:Sample.251.MySamples:sample2"))

p <- labkey.provenance.createProvenanceParams(name="step1", description="initial step",
  objectInputs=oi[["LSID"]], materialInputs=mi)
```

---

labkey.provenance.startRecording

*Start a provenance recording*

---

**Description**

Function to start a provenance recording session, if successful a provenance recording ID is returned which can be used to add additional steps to the provenance run. Note: this function is in beta and not yet final, changes should be expected so exercise caution when using it.

**Usage**

```
labkey.provenance.startRecording(baseUrl=NULL, folderPath, provenanceParams = NULL)
```

**Arguments**

<code>baseUrl</code>	a string specifying the baseUrl for the labkey server
<code>folderPath</code>	a string specifying the folderPath
<code>provenanceParams</code>	the provenance parameter object which contains the options to include in this recording step

**Details**

Function to start a provenance recording. A provenance recording can contain an arbitrary number of steps to create a provenance run, but `stopRecording` must be called to finish the recording and create the run. If successful this will return a recording ID which is needed for subsequent steps. This is a premium feature and requires the Provenance LabKey module to function correctly.

**Value**

The generated recording ID which can be used in subsequent steps (or queries that support provenance).

**Author(s)**

Karl Lum

**See Also**

[labkey.provenance.createProvenanceParams](#), [labkey.provenance.addRecordingStep](#), [labkey.provenance.stopRecording](#)

**Examples**

```
## create provenance params with material inputs and data outputs and start a recording
library(Rlabkey)

mi <- data.frame(lsid=c("urn:lsid:labkey.com:Sample.251.MySamples:sample1",
  "urn:lsid:labkey.com:Sample.251.MySamples:sample2"))
do <- data.frame(
  lsid="urn:lsid:labkey.com:AssayRunTSVData.Folder-251:12c70994-7ce5-1038-82f0-9c1487dbd334")

p <- labkey.provenance.createProvenanceParams(name="step1", description="initial step",
  materialInputs=mi, dataOutputs=do)
labkey.provenance.startRecording(baseUrl="https://labkey.org/labkey/",
  folderPath = "Provenance", provenanceParams=p)
```

---

labkey.provenance.stopRecording  
*Stop a provenance recording*

---

## Description

Function to end a provenance recording and create and save the provenance run on the server. Note: this function is in beta and not yet final, changes should be expected so exercise caution when using it.

## Usage

```
labkey.provenance.stopRecording(baseUrl=NULL, folderPath, provenanceParams = NULL)
```

## Arguments

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
provenanceParams	the provenance parameter object which contains the options to include in this recording step

## Details

Function to stop the provenance recording associated with the recording ID, this will create a provenance run using all the steps (with inputs and outputs) associated with the recording ID. The recording ID that was obtained from a previous startRecording function call must be passed into the provenanceParams config. This is a premium feature and requires the Provenance LabKey module to function correctly.

## Value

The serialized provenance run that was created.

## Author(s)

Karl Lum

## See Also

[labkey.provenance.createProvenanceParams](#), [labkey.provenance.startRecording](#), [labkey.provenance.addRecording](#)

## Examples

```
library(Rlabkey)

## object inputs (from an assay run) and material inputs
##
oi <- labkey.selectRows(baseUrl="https://labkey.org/labkey/", folderPath = "Provenance",
  schemaName="assay.General.titer",
  queryName="Data",
  colSelect= c("LSID"),
  colFilter=makeFilter(c("Run/RowId","EQUAL","253")))
mi <- data.frame(lsid=c("urn:lsid:labkey.com:Sample.251.MySamples:sample1",
  "urn:lsid:labkey.com:Sample.251.MySamples:sample2"))

p <- labkey.provenance.createProvenanceParams(name="step1", description="initial step",
  objectInputs=oi[["LSID"]], materialInputs=mi)
r <- labkey.provenance.startRecording(baseUrl="https://labkey.org/labkey/",
  folderPath = "Provenance", provenanceParams=p)
run <- labkey.provenance.stopRecording(baseUrl="https://labkey.org/labkey/",
  folderPath = "Provenance",
  provenanceParams=labkey.provenance.createProvenanceParams(name="final step",
  recordingId=r$recordingId))
```

---

labkey.query.import     *Bulk import an R data frame into a LabKey Server table using file import.*

---

## Description

Bulk import an R data frame into a LabKey Server table using file import.

## Usage

```
labkey.query.import(baseUrl, folderPath,
  schemaName, queryName, toImport, options = NULL)
```

## 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
toImport	a data frame containing rows of data to be imported
options	(optional) a list containing options specific to the import action of the query

## Details

This command mimics the "Import bulk data" option that you see in the LabKey server UI for a table/query. It takes the passed in `toImport` data frame and writes it to a temp file to be posted to the import action for the given LabKey query. It is very similar to the `labkey.importRows` command but will be much more performant.

Multiple rows of data can be imported in bulk using the `toImport` data frame. The names of the data in the data frame must be the column names from the LabKey Server.

LabKey data types support different import options. The list of valid options for each query will vary, but some common examples include:

- `insertOption` (string) : Whether the import action should be done as an insert, creating new rows for each provided row of the data frame, or a merge. When merging during import, any data you provide for the rows representing records that already exist will replace the previous values. Note that when updating an existing record, you only need to provide the columns you wish to update, existing data for other columns will be left as is. Available options are "INSERT" and "MERGE". Defaults to "INSERT".
- `auditBehavior` (string) : Set the level of auditing details for this import action. Available options are "SUMMARY" and "DETAILED". SUMMARY - Audit log reflects that a change was made, but does not mention the nature of the change. DETAILED - Provides full details on what change was made, including values before and after the change. Defaults to the setting as specified by the LabKey query.
- `importLookupByAlternateKey` (boolean) : Allows lookup target rows to be resolved by values rather than the target's primary key. This option will only be available for lookups that are configured with unique column information. Defaults to FALSE.

## Value

A list is returned with the row count for the number of affected rows. If options are provided, additional details may be included in the response object related to those options.

## Author(s)

Cory Nathe

## See Also

[labkey.insertRows](#), [labkey.updateRows](#), [labkey.importRows](#)

## Examples

```
## 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)

df <- data.frame(
```

```

    name=c("test1","test2","test3"),
    customInt=c(1:3),
    customString=c("aaa", "bbb", "ccc")
  )

  importedInfo <- labkey.query.import(
    "http://localhost:8080/labkey",
    folderPath="/apisamples", schemaName="samples", queryName="SampleType1",
    toImport=df, options=list(insertOption = "MERGE", auditBehavior = "DETAILED")
  )

  importedInfo$rowCount

```

---

labkey.rstudio.initReport

*Initialize a RStudio session for LabKey R report source editing*

---

## Description

LabKey-RStudio integration helper. Not intended for use outside RStudio.

## Usage

```
labkey.rstudio.initReport(apiKey = "", baseUrl = "", folderPath,
  reportEntityId, skipViewer = FALSE, skipEdit = FALSE)
```

## Arguments

apiKey	session key from your server
baseUrl	server location including context path, if any. e.g. <a href="https://www.labkey.org/">https://www.labkey.org/</a>
folderPath	a string specifying the folderPath
reportEntityId	LabKey report's entityId
skipViewer	(TRUE   FALSE) TRUE to skip setting up LabKey schema viewer in RStudio
skipEdit	(TRUE   FALSE) TRUE to open file in editor

## Examples

```

## RStudio console only
library(Rlabkey)
labkey.rstudio.initReport(apiKey="session|abcdef0123456789abcdef0123456789",
  baseUrl="http://labkey/", folderPath="home",
  reportEntityId="0123456a-789b-1000-abcd-01234567abcde")

```

---

labkey.rstudio.initRStudio  
*Initialize a RStudio session for LabKey integration*

---

**Description**

LabKey-RStudio integration helper. Not intended for use outside RStudio.

**Usage**

```
labkey.rstudio.initRStudio(apiKey = "", baseUrl = "", folderPath, skipViewer = FALSE)
```

**Arguments**

apiKey	session key from your server
baseUrl	server location including context path, if any. e.g. <a href="https://www.labkey.org/">https://www.labkey.org/</a>
folderPath	a string specifying the folderPath
skipViewer	(TRUE   FALSE) TRUE to skip setting up LabKey schema viewer in RStudio

**Examples**

```
## RStudio console only
library(Rlabkey)
labkey.rstudio.initRStudio(apiKey="session|abcdef0123456789abcdef0123456789",
  baseUrl="http://labkey/", folderPath="home")
```

---

labkey.rstudio.initSession  
*Initialize a RStudio session for LabKey integration using a time one request id*

---

**Description**

LabKey-RStudio integration helper. Not intended for use outside RStudio.

**Usage**

```
labkey.rstudio.initSession(requestId, baseUrl)
```

**Arguments**

requestId	A one time request id generated by LabKey server for initializing RStudio
baseUrl	server location including context path, if any. e.g. <a href="https://www.labkey.org/">https://www.labkey.org/</a>

## Examples

```
## RStudio console only
library(Rlabkey)
labkey.rstudio.initSession(requestId="a60228c8-9448-1036-a7c5-ab541dc15ee9",
  baseUrl="http://labkey/")
```

---

```
labkey.rstudio.isInitialized
  Check valid rlabkey session
```

---

## Description

LabKey-RStudio integration helper. Not intended for use outside RStudio.

## Usage

```
labkey.rstudio.isInitialized()
```

## Examples

```
## RStudio console only
library(Rlabkey)
labkey.rstudio.isInitialized()
```

---

```
labkey.rstudio.saveReport
  Update RStudio report source back to LabKey
```

---

## Description

LabKey-RStudio integration helper. Not intended for use outside RStudio.

## Usage

```
labkey.rstudio.saveReport(folderPath, reportEntityId, reportFilename,
  useWarning = FALSE)
```



**Arguments**

folderPath      a string specifying the folderPath  
 reportEntityId   LabKey report's entityId  
 reportFilename   The filename to save  
 useWarning      (TRUE | FALSE) TRUE to prompt user choices to save

**Examples**

```
## RStudio console only
library(Rlabkey)
labkey.rstudio.saveReport(folderPath="home",
  reportEntityId="0123456a-789b-1000-abcd-01234567abcde",
  reportFilename="knitrReport.Rhtml", useWarning=TRUE)
```

---

<code>labkey.saveBatch</code>	<i>Save an assay batch object to a labkey database</i>
-------------------------------	--

---

**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**

This function has been deprecated and will be removed in a future release, please use `labkey.experiment.saveBatch` instead as it supports the newer options for saving batch objects.

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/Documentation/wiki-page.view?name=createDatasetViaAssay>

**See Also**

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

**Examples**

```
## 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)

simpledf <- 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
)
```

---

labkey.security.createContainer

*Creates a new container, which may be a project, folder, or workbook, on the server*

---

### Description

Create a new container, which may be a project, folder, or workbook, on the LabKey server with parameters to control the containers name, title, description, and folder type.

### Usage

```
labkey.security.createContainer(baseUrl=NULL, parentPath, name = NULL, title = NULL,
  description = NULL, folderType = NULL, isWorkbook = FALSE)
```

### Arguments

baseUrl	A string specifying the baseUrl for the labkey server.
parentPath	A string specifying the parentPath for the new container.
name	The name of the container, required for projects or folders.
title	The title of the container, used primarily for workbooks.
description	The description of the container, used primarily for workbooks.
folderType	The name of the folder type to be applied (ex. Study or Collaboration).
isWorkbook	Whether this a workbook should be created. Defaults to false.

### Details

This function allows for users with proper permissions to create a new container, which may be a project, folder, or workbook, on the LabKey server with parameters to control the containers name, title, description, and folder type. If the container already exists or the user does not have permissions, an error message will be returned.

**Value**

Returns information about the newly created container.

**Author(s)**

Cory Nathe

**See Also**

[labkey.getFolders](#), [labkey.security.getContainers](#), [labkey.security.deleteContainer](#), [labkey.security.moveContainer](#)

**Examples**

```
library(Rlabkey)

labkey.security.createContainer(baseUrl="http://labkey/", parentPath = "/home",
  name = "NewFolder", description = "My new folder has this description",
  folderType = "Collaboration"
)
```

---

labkey.security.deleteContainer

*Deletes an existing container, which may be a project, folder, or workbook*

---

**Description**

Deletes an existing container, which may be a project, folder, or workbook, and all of its children from the Labkey server.

**Usage**

```
labkey.security.deleteContainer(baseUrl=NULL, folderPath)
```

**Arguments**

baseUrl	A string specifying the baseUrl for the labkey server.
folderPath	A string specifying the folderPath to be deleted.

**Details**

This function allows for users with proper permissions to delete an existing container, which may be a project, folder, or workbook, from the LabKey server. This will also remove all subfolders of the container being deleted. If the container does not exist or the user does not have permissions, an error message will be returned.

**Value**

Returns a success message for the container deletion action.

**Author(s)**

Cory Nathe

**See Also**

[labkey.getFolders](#), [labkey.security.getContainers](#), [labkey.security.createContainer](#), [labkey.security.moveContainer](#)

**Examples**

```
library(Rlabkey)
```

```
labkey.security.deleteContainer(baseUrl="http://labkey/", folderPath = "/home/FolderToDelete")
```

---

```
labkey.security.getContainers
```

*Returns information about the specified container*

---

**Description**

Returns information about the specified container, including the user's current permissions within that container. If the includeSubfolders config option is set to true, it will also return information about all descendants the user is allowed to see.

**Usage**

```
labkey.security.getContainers(baseUrl=NULL, folderPath, includeEffectivePermissions=TRUE,  
includeSubfolders=FALSE, depth=50)
```

**Arguments**

baseUrl	A string specifying the baseUrl for the labkey server.
folderPath	A string specifying the folderPath.
includeEffectivePermissions	If set to false, the effective permissions for this container resource will not be included (defaults to true).
includeSubfolders	If set to true, the entire branch of containers will be returned. If false, only the immediate children of the starting container will be returned (defaults to false).
depth	May be used to control the depth of recursion if includeSubfolders is set to true.

## Details

This function returns information about the specified container, including the user's current permissions within that container. If the includeSubfolders config option is set to true, it will also return information about all descendants the user is allowed to see. The depth of the results for the included subfolders can be controlled with the depth parameter.

## Value

The data frame containing the container properties for the current folder and subfolders, including name, title, id, path, type, folderType, and effectivePermissions.

## Author(s)

Cory Nathe

## See Also

[labkey.getFolders](#), [labkey.security.createContainer](#), [labkey.security.deleteContainer](#), [labkey.security.moveContainer](#)

## Examples

```
library(Rlabkey)

labkey.security.getContainers(
  baseUrl="http://labkey/", folderPath = "home",
  includeEffectivePermissions = FALSE, includeSubfolders = TRUE, depth = 2
)
```

---

labkey.security.impersonateUser  
*Start impersonating a user*

---

## Description

For site-admins or project-admins only, start impersonating a user based on the userId or email address.

## Usage

```
labkey.security.impersonateUser(baseUrl=NULL, folderPath,
  userId=NULL, email=NULL)
```

### Arguments

baseUrl	A string specifying the baseUrl for the LabKey server.
folderPath	A string specifying the folderPath in which to impersonate the user.
userId	The id of the user to be impersonated. Either this or email is required.
email	The email of the user to be impersonated. Either this or userID is required.

### Details

Admins may impersonate other users to perform actions on their behalf. Site admins may impersonate any user in any project. Project admins must execute this command in a project in which they have admin permission and may impersonate only users that have access to the project.

To finish an impersonation session use [labkey.security.stopImpersonating](#).

### Value

Returns a success message based on a call to [labkey.whoAmI](#).

### Author(s)

Cory Nathe

### See Also

[labkey.whoAmI](#), [labkey.security.stopImpersonating](#)

### Examples

```
library(Rlabkey)

labkey.security.impersonateUser(baseUrl="http://labkey/", folderPath = "/home",
  email = "reader@localhost.test"
)
```

---

labkey.security.moveContainer

*Moves an existing container, which may be a folder or workbook*

---

### Description

Moves an existing container, which may be a folder or workbook, to be the subfolder of another folder and/or project on the LabKey server.

**Usage**

```
labkey.security.moveContainer(baseUrl=NULL, folderPath,  
    destinationParent, addAlias = TRUE)
```

**Arguments**

<code>baseUrl</code>	A string specifying the baseUrl for the labkey server.
<code>folderPath</code>	A string specifying the folderPath to be moved. Additionally, the container entity id is also valid.
<code>destinationParent</code>	The container path of destination parent. Additionally, the destination parent entity id is also valid.
<code>addAlias</code>	Add alias of current container path to container that is being moved (defaults to true).

**Details**

This function moves an existing container, which may be a folder or workbook, to be the subfolder of another folder and/or project on the LabKey server. Projects and the root container can not be moved. If the target or destination container does not exist or the user does not have permissions, an error message will be returned.

**Value**

Returns a success message for the container move action with the new path.

**Author(s)**

Cory Nathe

**See Also**

[labkey.getFolders](#), [labkey.security.getContainers](#), [labkey.security.createContainer](#), [labkey.security.deleteContainer](#)

**Examples**

```
library(Rlabkey)  
  
labkey.security.moveContainer(baseUrl="http://labkey/", folderPath = "/home/FolderToMove",  
    destinationParent = "/OtherProject", addAlias = TRUE  
)
```



---

`labkey.security.stopImpersonating`  
*Stop impersonating a user*

---

**Description**

Stop impersonating a user while keeping the original user logged in.

**Usage**

```
labkey.security.stopImpersonating(baseUrl=NULL)
```

**Arguments**

`baseUrl`            A string specifying the `baseUrl` for the LabKey server.

**Details**

If you are currently impersonating a user in this session, you can use this function to stop the impersonation and return back to the original user logged in.

To start an impersonation session use [labkey.security.impersonateUser](#).

**Value**

Returns a success message based on a call to [labkey.whoAmI](#).

**Author(s)**

Cory Nathe

**See Also**

[labkey.whoAmI](#), [labkey.security.impersonateUser](#)

**Examples**

```
library(Rlabkey)

labkey.security.stopImpersonating(baseUrl="http://labkey/")
```

---

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 = NULL, folderPath, schemaName, queryName,
  viewName = NULL, colSelect = NULL, maxRows = NULL,
  rowOffset = NULL, colSort = NULL, colFilter = NULL,
  showHidden = FALSE, colNameOpt="caption",
  containerFilter = NULL, parameters = NULL,
  includeDisplayValues = FALSE, method = "POST")
```

### 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.
colSelect	(optional) a vector of strings specifying which columns of a dataset or view to import. <ul style="list-style-type: none"> <li>• The wildcard character ("*") may also be used here to get all columns including those not in the default view.</li> <li>• If you include a column that is a lookup (foreign key) the value of the primary key for that target will be returned.</li> <li>• Use a backslash character ("/") to include non-primary key columns from a lookup target (foreign key), e.g "LookupColumnName/targetColumn".</li> <li>• When using a string to specify the colSelect set, the column names must be separated by a comma and not include spaces between the names.</li> </ul>
maxRows	(optional) an integer specifying how many rows of data 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, the retrieval starts with the first row.
colSort	(optional) a string including the name of the column to sort preceded by a "+" or "-" to indicate sort direction
colFilter	(optional) a vector or array object created by the makeFilter function which contains the column name, operator and value of the filter(s) to be applied to the retrieved data.

showHidden	(optional) a logical value indicating whether or not to return data columns that would normally be hidden from user view. Defaults to FALSE if no value provided.
colNameOpt	(optional) controls the name source for the columns of the output dataframe, with valid values of 'caption', 'fieldname', and 'rname'
containerFilter	(optional) Specifies the containers to include in the scope of selectRows request. A value of NULL is equivalent to "Current". Valid values are <ul style="list-style-type: none"> <li>• "Current": Include the current folder only</li> <li>• "CurrentAndSubfolders": Include the current folder and all subfolders</li> <li>• "CurrentPlusProject": Include the current folder and the project that contains it</li> <li>• "CurrentAndParents": Include the current folder and its parent folders</li> <li>• "CurrentPlusProjectAndShared": Include the current folder plus its project plus any shared folders</li> <li>• "AllFolders": Include all folders for which the user has read permission</li> </ul>
parameters	(optional) List of name/value pairs for the parameters if the SQL references underlying queries that are parameterized. For example, parameters=c("X=1", "Y=2").
includeDisplayValues	(optional) a logical value indicating if display values should be included in the response object for lookup fields.
method	(optional) HTTP method to use for the request, defaults to POST.

## 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 lower casing the entire name. This option is the way a data frame is passed to a script running in a 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/Documentation/wiki-page.view?name=savingViews>

### See Also

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

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

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

### Examples

```
## select from a list named AllTypes
# library(Rlabkey)

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="Lists", queryName="AllTypes",
  viewName="rowbyrow")

## select a subset of columns
colSelect=c("TextFld", "IntFld")
subsetcols <- labkey.selectRows(baseUrl="http://localhost:8080/labkey",
  folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
  colSelect=colSelect)

## including columns from a lookup (foreign key) field
lookupcols <- labkey.selectRows(baseUrl="http://localhost:8080/labkey",
```

```
folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
colSelect="TextFld,IntFld,IntFld/LookupValue")
```

---

labkey.setCurlOptions *Modify the current set of Curl options that are being used in the existing session*

---

### Description

Rlabkey uses the package htr to connect to the LabKey Server.

### Arguments

options args	a variable list of arguments to set the RCurl options
ssl_verifyhost	check the existence of a common name and also verify that it matches the host-name provided
ssl_verifypeer	specifies whether curl will verify the peer's certificate
followlocation	specify is curl should follow any location header that is sent in the HTTP request
sslversion	the SSL version to use

### Details

This topic explains how to configure Rlabkey to work with a LabKey Server running SSL.

Rlabkey uses the package htr to connect to the LabKey Server. On Windows, the htr package is not configured for SSL by default. In order to connect to a HTTPS enabled LabKey Server, you will need to perform the following steps:

1. Create or download a "ca-bundle" file.

We recommend using ca-bundle file that is published by Mozilla. See <http://curl.haxx.se/docs/caextract.html>. You have two options:

Download the ca-bundle.crt file from the link named "HTTPS from github:" on <http://curl.haxx.se/docs/caextract.html>  
 Create your own ca-bundle.crt file using the instructions provided on <http://curl.haxx.se/docs/caextract.html>

2. Copy the ca-bundle.crt file to a location on your hard-drive.  
 If you will be the only person using the Rlabkey package on your computer, we recommend that you

create a directory named 'labkey' in your home directory  
 copy the ca-bundle.crt into the 'labkey' directory

If you are installing this file on a server where multiple users will use may use the Rlabkey package, we recommend that you create a directory named 'c:labkey'

copy the ca-bundle.crt into the 'c:labkey' directory

3. Create a new Environment variable named 'RLABKEY\_CAINFO\_FILE'

On Windows 7, Windows Server 2008 and earlier

Select Computer from the Start menu.

Choose System Properties from the context menu.

Click Advanced system settings > Advanced tab.

Click on Environment Variables.

Under System Variables click on the new button.

For Variable Name: enter RLABKEY\_CAINFO\_FILE

For Variable Value: enter the path of the ca-bundle.crt you created above.

Hit the Ok buttons to close all the windows.

On Windows 8, Windows 2012 and above

Drag the Mouse pointer to the Right bottom corner of the screen.

Click on the Search icon and type: Control Panel.

Click on -> Control Panel -> System and Security.

Click on System -> Advanced system settings > Advanced tab.

In the System Properties Window, click on Environment Variables.

Under System Variables click on the new button.

For Variable Name: enter RLABKEY\_CAINFO\_FILE

For Variable Value: enter the path of the ca-bundle.crt you created above.

Hit the Ok buttons to close all the windows.

Now you can start R and begin working.

This command can also be used to provide an alternate location / path to your .netrc file. Example:  
labkey.setCurloptions(NETRC\_FILE = '/path/to/alternate/\_netrc')

---

labkey.setDebugMode     *Helper function to enable/disable debug mode.*

---

## Description

When debug mode is enabled, the GET/POST calls with output information about the request being made and will output a raw string version of the response object.

## Usage

```
labkey.setDebugMode(debug = FALSE)
```

**Arguments**

debug            a boolean specifying if debug mode is enabled or disabled

**Author(s)**

Cory Nathe

**Examples**

```
library(Rlabkey)
labkey.setDebugMode(TRUE)
labkey.executeSql(
  baseUrl="http://localhost:8080/labkey",
  folderPath="/home",
  schemaName="core",
  sql = "select * from containers")
```

---

labkey.setDefaults     *Set the default parameters used for all http or https requests*

---

**Description**

Use this function to set the default baseUrl and authentication parameters as package environment variables to be used for all http or https requests. You can also use labkey.setDefaults() without any parameters to reset/clear these settings.

**Usage**

```
labkey.setDefaults(apiKey="", baseUrl="", email="", password="")
```

**Arguments**

apiKey            session key from your server  
baseUrl           server location including context path, if any. e.g. <https://www.labkey.org/>  
email             user email address  
password          user password

**Details**

Note: Support for API keys was added in LabKey Server release 16.2; they are not supported in 16.1 or earlier.

An API key can be used to authorize Rlabkey functions that access protected content on LabKey Server. Using an API key avoids copying and storing credentials on the client machine. Also, all Rlabkey access is tied to the current browser session, which means the code runs in the same context as the browser (e.g. same user, same authorizations, same declared terms of use and PHI level, same impersonation state, etc.).

A site administrator must first enable the use of API keys on that LabKey Server. Once enabled, any logged in user can generate an API key by clicking their display name (upper right) and selecting "API Keys". The API Key page creates and displays keys that can be copied and pasted into a labkey.setDefaults() statement to tie an Rlabkey session to the authorization and session information already set in the browser.

If an API key is not provided, you can also use this function for basic authentication via email and password. Note that both email and password must be set via a labkey.setDefaults() call. If an apiKey is also set, that will be given preference and the email/password will not be used for authentication. Once authenticated via email/password, you can make multiple labkey.get or labkey.post API calls using that same connection.

**Examples**

```
## Example of setting and clearing an API key and/or email/password.
# library(Rlabkey)

labkey.setDefaults(apiKey="session|abcdef0123456789abcdef0123456789")
labkey.setDefaults(email="testing@localhost.test", password="password")

## Functions invoked at this point share authorization
## and session information with the browser session

labkey.setDefaults() # called without any parameters will reset/clear the environment variables
```

---

```
labkey.setModuleProperty
    Set module property value
```

---

**Description**

Set module property value for a specific folder or as site wide (with folderPath '/')



**Usage**

```
labkey.setModuleProperty(baseUrl=NULL, folderPath, moduleName, propName, propValue)
```

**Arguments**

baseUrl	server location including context path, if any. e.g. https://www.labkey.org/
folderPath	a string specifying the folderPath
moduleName	name of the module
propName	The module property name
propValue	The module property value to save

**Examples**

```
library(Rlabkey)
labkey.setModuleProperty(baseUrl="http://labkey/", folderPath="flowProject",
  moduleName="flow", propName="ExportToScriptFormat", propValue="zip")
```

---

```
labkey.transform.getRunPropertyValue
```

*Assay transform script helper function to get a run property value from a data.frame*

---

**Description**

A function that takes in data.frame of the run properties info for a given assay transform script execution and returns the value for a given property name.

**Usage**

```
labkey.transform.getRunPropertyValue(runProps, propName)
```

**Arguments**

runProps	the data.frame of the run property key/value pairs
propName	the name of the property to get the value of within the runProps data.frame

**Details**

This helper function will most likely be used within an assay transform script after the labkey.transform.readRunPropertiesFile function has been called to load the full set of run properties.

## Examples

```
# library(Rlabkey)

run.props = labkey.transform.readRunPropertiesFile("${runInfo}");
run.data.file = labkey.transform.getRunPropertyValue(run.props, "runDataFile");
```

---

labkey.transform.readRunPropertiesFile

*Assay transform script helper function to read a run properties file*

---

## Description

A function that takes in the full path to the LabKey generated run properties file and returns a data.frame of the key value pairs for the lines within that file. This helper function would be used as part of an assay transform script written in R and associated with an assay design.

## Usage

```
labkey.transform.readRunPropertiesFile(runInfoPath)
```

## Arguments

runInfoPath      the full file system path to the generated run properties file

## Details

The most common scenario is that the assay transform script will get the run properties file path added into the running script as a replacement variable. To use that replacement variable for this helper function, you can pass in the runInfoPath parameter as "\$runInfo".

## Examples

```
# library(Rlabkey)

labkey.transform.readRunPropertiesFile("${runInfo}")
```

---

labkey.truncateTable *Delete all rows from a table*

---

### Description

Delete all rows from the specified table.

### Usage

```
labkey.truncateTable(baseUrl = NULL, 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 name of the schema of the domain
queryName	a string specifying the query name

### Details

Deletes all rows in the table in a single transaction and will also log a single audit event for the action. Not all tables support truncation, if a particular table doesn't support the action, an error will be returned. The current list of tables supporting truncation include : lists, datasets, issues, sample sets, data classes.

### Value

Returns the count of the number of rows deleted.

### Author(s)

Karl Lum

### See Also

[labkey.deleteRows](#)

### Examples

```
## create a data frame and infer it's fields
library(Rlabkey)

labkey.truncateTable(baseUrl="http://labkey/", folderPath="home",
  schemaName="lists", queryName="people")
```

---

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, provenanceParams=NULL)
```

### 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
provenanceParams	the provenance parameter object which contains the options to include as part of a provenance recording. This is a premium feature and requires the Provenance LabKey module to function correctly, if it is not present this parameter will be ignored.

### 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.importRows](#),  
[labkey.deleteRows](#), [labkey.query.import](#), [labkey.provenance.createProvenanceParams](#),  
[labkey.provenance.startRecording](#), [labkey.provenance.addRecordingStep](#), [labkey.provenance.stopRecording](#)

**Examples**

```
## 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", queryName="AllTypes",
  toInsert=newrow)
newRowId <- insertedRow$rows[[1]]$RowId

selectedRow<-labkey.selectRows("http://localhost:8080/labkey",
  folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
  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", queryName="AllTypes",
  toUpdate=updaterow)
selectedRow<-labkey.selectRows("http://localhost:8080/labkey",
  folderPath="/apisamples", schemaName="lists", queryName="AllTypes",
  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", queryName="AllTypes",
  toDelete=deleterow)
str(result)
```

---

labkey.webdav.delete *Deletes the provided file/folder on a LabKey Server via WebDAV*

---

### Description

This will delete the supplied file or folder under the specified LabKey Server project using WebDAV.

### Usage

```
labkey.webdav.delete(
  baseUrl=NULL,
  folderPath,
  remoteFilePath,
  fileSet='@files'
)
```

### Arguments

**baseUrl** a string specifying the baseUrl for the labkey server

**folderPath** a string specifying the folderPath

**remoteFilePath** the path to delete, relative to the LabKey folder root.

**fileSet** (optional) the name of file server fileSet, which is typically "@files" (the default value for this argument). In some cases this might be "@pipeline" or "@fileset".

### Details

This will delete the supplied file or folder under the specified LabKey Server project using WebDAV. Note: if a folder is provided, it will delete that folder and contents.

### Value

TRUE if the folder was deleted successfully

### Author(s)

Ben Bimber, Ph.D.

**See Also**

[labkey.webdav.get](#), [labkey.webdav.put](#), [labkey.webdav.mkdir](#), [labkey.webdav.mkdir](#), [labkey.webdav.listdir](#), [labkey.webdav.pathExists](#), [labkey.webdav.downloadFolder](#)

**Examples**

```
library(Rlabkey)

#delete an entire directory and contents
labkey.webdav.delete(baseUrl="http://labkey/", folderPath="home", remoteFilePath="folder1")

#delete single file
labkey.webdav.delete(baseUrl="http://labkey/", folderPath="home", remoteFilePath="folder/file.txt")
```

---

labkey.webdav.downloadFolder

*Recursively download a folder via WebDAV*

---

**Description**

This will recursively download a folder from a LabKey Server using WebDAV.

**Usage**

```
labkey.webdav.downloadFolder(  
  localBaseDir,  
  baseUrl=NULL,  
  folderPath,  
  remoteFilePath,  
  overwriteFiles=TRUE,  
  mergeFolders=TRUE,  
  fileSet='@files'  
)
```

**Arguments**

localBaseDir	the local filepath where this directory will be saved. a subfolder with the remote directory name will be created.
baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
remoteFilePath	the path of this folder on the remote server, relative to the folder root.
overwriteFiles	(optional) if true, any pre-existing file at this location will be overwritten. Defaults to TRUE

mergeFolders	(optional) if false, any pre-existing local folders in the target location will be deleted if there is an incoming folder of the same name. If true, these existing folders will be left alone, and remote files downloaded into them. Existing file conflicts will be handled based on the overwriteFiles parameter. Defaults to TRUE
fileSet	(optional) the name of file server fileSet, which is typically "@files" (the default value for this argument). In some cases this might be "@pipeline" or "@fileset".

### Details

This will recursively download a folder from a LabKey Server using WebDAV. This is essentially a wrapper that recursively calls labkey.webdav.get to download all files in the remote folder.

### Value

TRUE or FALSE, depending on if this folder was successfully downloaded

### Author(s)

Ben Bimber, Ph.D.

### See Also

[labkey.webdav.get](#), [labkey.webdav.put](#), [labkey.webdav.mkDir](#), [labkey.webdav.mkDirs](#), [labkey.webdav.pathExists](#), [labkey.webdav.listdir](#), [labkey.webdav.delete](#)

### Examples

```
## download folder from a LabKey Server
library(Rlabkey)

labkey.webdav.downloadFolder(baseUrl="http://labkey/",
  folderPath="home",
  remoteFilePath="folder1",
  localBaseDir="destFolder",
  overwrite=TRUE
)
```

---

labkey.webdav.get      *Download a file via WebDAV*

---

### Description

This will download a file from a LabKey Server using WebDAV.



## Usage

```
labkey.webdav.get(  
  baseUrl=NULL,  
  folderPath,  
  remoteFilePath,  
  localFilePath,  
  overwrite=TRUE,  
  fileSet='@files'  
)
```

## Arguments

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
remoteFilePath	the path of this file on the remote server, relative to the folder root.
localFilePath	the local filepath where this file will be saved
overwrite	(optional) if true, any pre-existing file at this location will be overwritten. Defaults to TRUE
fileSet	(optional) the name of file server fileSet, which is typically "@files" (the default value for this argument). In some cases this might be "@pipeline" or "@fileset".

## Details

Download a single file from a LabKey Server to the local machine using WebDAV.

## Value

TRUE or FALSE, depending on if this file was downloaded and exists locally. Will return FALSE if the already file exists and overwrite=F.

## Author(s)

Ben Bimber, Ph.D.

## See Also

[labkey.webdav.put](#), [labkey.webdav.mkdir](#), [labkey.webdav.mkdirs](#), [labkey.webdav.pathExists](#), [labkey.webdav.listdir](#), [labkey.webdav.delete](#), [labkey.webdav.downloadFolder](#)

## Examples

```
## download a single file from a LabKey Server  
library(Rlabkey)
```

```
labkey.webdav.get(  
  baseUrl="http://labkey/",  
  folderPath="home",
```

```
remoteFilePath="folder/myFile.txt",
localFilePath="myDownloadedFile.txt",
overwrite=TRUE
)
```

---

labkey.webdav.listDir *List the contents of a LabKey Server folder via WebDAV*

---

### Description

This will list the contents of a LabKey Server folder using WebDAV.

### Usage

```
labkey.webdav.listDir(
  baseUrl=NULL,
  folderPath,
  remoteFilePath,
  fileSet='@files',
  haltOnError=TRUE
)
```

### Arguments

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
remoteFilePath	path of the folder on the remote server, relative to the folder root.
fileSet	(optional) the name of file server fileSet, which is typically "@files" (the default value for this argument). In some cases this might be "@pipeline" or "@fileset".
haltOnError	(optional) Specifies whether this request should fail if the requested path does not exist. Defaults to TRUE

### Details

Lists the contents of a folder on a LabKey Server using WebDAV.

### Value

A list with each item under this folder. Each item (file or directory) is a list with the following attributes:

- "files": A list of the files, where each has the following attributes:
  - "id": The relative path to this item, not encoded
  - "href": The relative URL to this item, HTML encoded

- "text": A dataset in a date based study
- "creationdate": The date this item was created
- "createdby": The user that created this file
- "lastmodified": The last modification time
- "contentlength": The content length
- "size": The file size
- "isdirectory": TRUE or FALSE, depending on whether this item is a directory
- "fileCount": If this item is a directory, this property will be present, listing the the total files in this location

**Author(s)**

Ben Bimber, Ph.D.

**See Also**

[labkey.webdav.get](#), [labkey.webdav.put](#), [labkey.webdav.mkDir](#), [labkey.webdav.mkDirs](#), [labkey.webdav.pathExists](#),  
[labkey.webdav.delete](#), [labkey.webdav.downloadFolder](#)

**Examples**

```
library(Rlabkey)

json <- labkey.webdav.listDir(
  baseUrl="http://labkey/",
  folderPath="home",
  remoteFilePath="myFolder"
)
```

---

labkey.webdav.mkDir    *Create a folder via WebDAV*

---

**Description**

This will create a folder under the specified LabKey Server project using WebDAV.

**Usage**

```
labkey.webdav.mkDir(
  baseUrl=NULL,
  folderPath,
  remoteFilePath,
  fileSet='@files'
)
```

**Arguments**

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
remoteFilePath	the folder path to create, relative to the LabKey folder root.
fileSet	(optional) the name of file server fileSet, which is typically "@files" (the default value for this argument). In some cases this might be "@pipeline" or "@fileset".

**Details**

Creates a folder on a LabKey Server using WebDAV. If the parent directory does not exist, this will fail (similar to mkdir on linux)

**Value**

TRUE if the folder was created successfully

**Author(s)**

Ben Bimber, Ph.D.

**See Also**

[labkey.webdav.get](#), [labkey.webdav.put](#), [labkey.webdav.mkDirs](#), [labkey.webdav.pathExists](#), [labkey.webdav.listdir](#), [labkey.webdav.delete](#), [labkey.webdav.downloadFolder](#)

**Examples**

```
library(Rlabkey)

labkey.webdav.mkDir(
  baseUrl="http://labkey/",
  folderPath="home",
  remoteFilePath="toCreate"
)
```

---

labkey.webdav.mkDirs *Create a folder via WebDAV*

---

**Description**

This will create folder(s) under the specified LabKey Server project using WebDAV.

**Usage**

```
labkey.webdav.mkDirs(  
  baseUrl=NULL,  
  folderPath,  
  remoteFilePath,  
  fileSet='@files'  
)
```

**Arguments**

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
remoteFilePath	the folder path to create, relative to the LabKey folder root.
fileSet	(optional) the name of file server fileSet, which is typically "@files" (the default value for this argument). In some cases this might be "@pipeline" or "@fileset".

**Details**

Creates a folder on a LabKey Server using WebDAV. If the parent directory or directories do not exist, these will also be created (similar to mkdir -p on linux)

**Value**

TRUE if the folder was created successfully

**Author(s)**

Ben Bimber, Ph.D.

**See Also**

[labkey.webdav.get](#), [labkey.webdav.put](#), [labkey.webdav.mkdir](#), [labkey.webdav.pathExists](#), [labkey.webdav.listdir](#), [labkey.webdav.delete](#), [labkey.webdav.downloadFolder](#)

**Examples**

```
library(Rlabkey)  
  
labkey.webdav.mkDirs(  
  baseUrl="http://labkey/",  
  folderPath="home",  
  remoteFilePath="folder1/folder2/toCreate"  
)
```

labkey.webdav.pathExists

*Tests if a path exists on a LabKey Server via WebDAV*

---

### Description

This will test if the supplied file/folder exists folder under the specified LabKey Server project using WebDAV.

### Usage

```
labkey.webdav.pathExists(  
    baseUrl=NULL,  
    folderPath,  
    remoteFilePath,  
    fileSet='@files'  
)
```

### Arguments

baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
remoteFilePath	the path to test, relative to the LabKey folder root.
fileSet	(optional) the name of file server fileSet, which is typically "@files" (the default value for this argument). In some cases this might be "@pipeline" or "@fileset".

### Details

This will test if the supplied file/folder exists folder under the specified LabKey Server project using WebDAV.

### Value

TRUE if the folder was created successfully

### Author(s)

Ben Bimber, Ph.D.

### See Also

[labkey.webdav.get](#), [labkey.webdav.put](#), [labkey.webdav.mkdir](#), [labkey.webdav.mkdirs](#), [labkey.webdav.listdir](#), [labkey.webdav.delete](#), [labkey.webdav.downloadFolder](#)

## Examples

```
library(Rlabkey)

# Test folder
labkey.webdav.pathExists(
  baseUrl="http://labkey/",
  folderPath="home",
  remoteFilePath="pathToTest"
)

# Test file
labkey.webdav.pathExists(
  baseUrl="http://labkey/",
  folderPath="home",
  remoteFilePath="folder/fileToTest.txt"
)
```

---

labkey.webdav.put      *Upload a file via WebDAV*

---

## Description

This will upload a file to a LabKey Server using WebDAV.

## Usage

```
labkey.webdav.put(
  localFile,
  baseUrl=NULL,
  folderPath,
  remoteFilePath,
  fileSet='@files',
  description=NULL
)
```

## Arguments

localFile	the local filepath to upload
baseUrl	a string specifying the baseUrl for the labkey server
folderPath	a string specifying the folderPath
remoteFilePath	the destination path of this file on the remote server, relative to the folder root.
fileSet	(optional) the name of file server fileSet, which is typically "@files" (the default value for this argument). In some cases this might be "@pipeline" or "@fileset".
description	(optional) the description to attach to this file on the remote server.

**Details**

Upload a single file from the local machine to a LabKey Server using WebDAV.

**Value**

TRUE if the file was uploaded successfully

**Author(s)**

Ben Bimber, Ph.D.

**See Also**

[labkey.webdav.get](#), [labkey.webdav.mkDir](#), [labkey.webdav.mkDirs](#), [labkey.webdav.pathExists](#), [labkey.webdav.listdir](#), [labkey.webdav.delete](#), [labkey.webdav.downloadFolder](#)

**Examples**

```
## upload a single file to a LabKey Server
library(Rlabkey)

labkey.webdav.put(
  localFile="myFileToUpload.txt",
  baseUrl="http://labkey/",
  folderPath="home",
  remoteFilePath="myFileToUpload.txt"
)
```

---

labkey.whoAmI

*Call the whoami API*

---

**Description**

Call the whoami API to get information about the current LabKey user.

**Usage**

```
labkey.whoAmI(baseUrl=NULL)
```

**Arguments**

`baseUrl` A string specifying the baseUrl for the LabKey server.

**Value**

Returns information about the logged in user including: `displayName`, `id`, `email`, and whether or not the user is impersonated.



**Author(s)**

Cory Nathe

**See Also**

[labkey.security.impersonateUser](#), [labkey.security.stopImpersonating](#)

**Examples**

```
library(Rlabkey)
labkey.whoAmI(baseUrl="http://labkey/")
```

---

IsFolders

*List the available folder paths*

---

**Description**

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

**Usage**

```
IsFolders(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 project root. These values can be set on a session using `curFolder<-`

**Author(s)**

Peter Hussey

**References**

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

**See Also**

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

**Examples**

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

lks<- getSession("https://www.labkey.org", "/home")

#returns values "/home" , "/home/_menus" , ...
lsFolders(lks)
```

---

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**

baseUrl            a string specifying the baseUrl for the LabKey Server, of the form http://<server dns name>/<contextroot>

**Details**

List the projects available at a given LabKey Server address.

**Value**

A character array containing the available projects, 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](#), [lsFolders](#), [lsSchemas](#)

**Examples**

```
## get list of projects on server, connect a session in one project,
## then list the folders in that project
# library(Rlabkey)

lsProjects("https://www.labkey.org")

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

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

---

lsSchemas

*List the available schemas*

---

**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**

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

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

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

---

**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  
DATE\_EQUAL  
NOT\_EQUAL  
DATE\_NOT\_EQUAL  
NOT\_EQUAL\_OR\_MISSING  
GREATER\_THAN  
DATE\_GREATER\_THAN  
LESS\_THAN  
DATE\_LESS\_THAN  
GREATER\_THAN\_OR\_EQUAL  
DATE\_GREATER\_THAN\_OR\_EQUAL  
LESS\_THAN\_OR\_EQUAL  
DATE\_LESS\_THAN\_OR\_EQUAL  
STARTS\_WITH  
DOES\_NOT\_START\_WITH  
CONTAINS  
DOES\_NOT\_CONTAIN  
CONTAINS\_ONE\_OF  
CONTAINS\_NONE\_OF  
IN  
NOT\_IN  
BETWEEN  
NOT\_BETWEEN  
MEMBER\_OF  
MISSING  
NOT\_MISSING  
MV\_INDICATOR  
NO\_MV\_INDICATOR  
Q  
ONTOLOGY\_IN\_SUBTREE  
ONTOLOGY\_NOT\_IN\_SUBTREE

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

**See Also**

[labkey.selectRows](#)

**Examples**

```
# library(Rlabkey)

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

## Using "in" operator:
makeFilter(c("RowId", "IN", "2;3;6"))

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

---

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

```
# library(Rlabkey)
RlabkeyUsersGuide()
```

---

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

---

## Description

A wrapper function to `labkey.saveBatch` which uses a session object and provides defaults for the Batch/Run names.

## Usage

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

## Arguments

<code>session</code>	the session key returned from <code>getSession</code>
<code>assayName</code>	a string specifying the name of the assay instance
<code>resultDataFrame</code>	a data frame containing rows of data to be inserted
<code>batchPropertyList</code>	a list of batch Properties
<code>runPropertyList</code>	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**

```
## 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")
sobj <- getSchema(s, "lists")
simpledf <- getRows(s, sobj$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<- saveResults(s, "SimpleMeans", results,
  batchPropertyList=bpl, runPropertyList=rpl)
```



# Index

## \* IO

- labkey.deleteRows, [12](#)
- labkey.domain.create, [14](#)
- labkey.domain.createAndLoad, [17](#)
- labkey.domain.createConditionalFormat, [19](#)
- labkey.domain.createConditionalFormatQueryFilter, [20](#)
- labkey.domain.createDesign, [22](#)
- labkey.domain.createIndices, [23](#)
- labkey.domain.drop, [24](#)
- labkey.domain.FILTER\_TYPES, [25](#)
- labkey.domain.get, [26](#)
- labkey.domain.inferFields, [27](#)
- labkey.domain.save, [28](#)
- labkey.executeSql, [29](#)
- labkey.experiment.createData, [31](#)
- labkey.experiment.createMaterial, [32](#)
- labkey.experiment.createRun, [33](#)
- labkey.experiment.SAMPLE\_DERIVATION\_PROTOCOL, [35](#)
- labkey.experiment.saveBatch, [36](#)
- labkey.experiment.saveRuns, [38](#)
- labkey.getDefaultViewDetails, [40](#)
- labkey.getFolders, [41](#)
- labkey.getLookupDetails, [42](#)
- labkey.getQueries, [45](#)
- labkey.getQueryDetails, [46](#)
- labkey.getQueryViews, [48](#)
- labkey.getRequestOptions, [50](#)
- labkey.getSchemas, [51](#)
- labkey.importRows, [52](#)
- labkey.insertRows, [53](#)
- labkey.pipeline.getFileStatus, [57](#)
- labkey.pipeline.getPipelineContainer, [58](#)
- labkey.pipeline.getProtocols, [59](#)
- labkey.pipeline.startAnalysis, [61](#)
- labkey.provenance.addRecordingStep, [62](#)
- labkey.provenance.createProvenanceParams, [63](#)
- labkey.provenance.startRecording, [65](#)
- labkey.provenance.stopRecording, [67](#)
- labkey.query.import, [68](#)
- labkey.saveBatch, [73](#)
- labkey.security.createContainer, [75](#)
- labkey.security.deleteContainer, [76](#)
- labkey.security.getContainers, [77](#)
- labkey.security.impersonateUser, [78](#)
- labkey.security.moveContainer, [79](#)
- labkey.security.stopImpersonating, [81](#)
- labkey.selectRows, [82](#)
- labkey.setDebugMode, [86](#)
- labkey.truncateTable, [91](#)
- labkey.updateRows, [92](#)
- labkey.webdav.delete, [94](#)
- labkey.webdav.downloadFolder, [95](#)
- labkey.webdav.get, [96](#)
- labkey.webdav.listDir, [98](#)
- labkey.webdav.mkDir, [99](#)
- labkey.webdav.mkDirs, [100](#)
- labkey.webdav.pathExists, [102](#)
- labkey.webdav.put, [103](#)
- labkey.whoAmI, [104](#)

## \* file

- getFolderPath, [5](#)
- getLookups, [6](#)
- getRows, [7](#)
- getSchema, [8](#)
- getSession, [10](#)

- labkey.makeRemotePath, 56
- lsFolders, 105
- lsProjects, 106
- lsSchemas, 107
- makeFilter, 108
- RlabkeyUsersGuide, 110
- saveResults, 111
- \* **package**
  - Rlabkey-package, 3
- getFolderPath, 5
- getLookups, 6, 8, 11, 112
- getRows, 6, 7, 11, 30, 83, 112
- getSchema, 6, 8, 8, 11, 112
- getSession, 5, 6, 8, 9, 10, 106–108, 112
- labkey.acceptSelfSignedCerts, 11
- labkey.curlOptions, 12
- labkey.deleteRows, 5, 12, 30, 41, 43, 46, 48, 49, 51, 53, 54, 74, 84, 91, 93
- labkey.domain.create, 14, 20–22, 24–26, 28, 29
- labkey.domain.createAndLoad, 17
- labkey.domain.createConditionalFormat, 16, 19, 21, 22, 25, 29
- labkey.domain.createConditionalFormatQueryFilter, 16, 20, 20, 22, 25, 29
- labkey.domain.createDesign, 16, 18, 20, 21, 22, 24–26, 28, 29
- labkey.domain.createIndices, 16, 18, 22, 23, 24, 26, 28, 29
- labkey.domain.drop, 16, 18, 20–22, 24, 24, 25, 26, 28, 29
- labkey.domain.FILTER\_TYPES, 16, 20–22, 25, 29
- labkey.domain.get, 16, 18, 20–22, 24, 25, 26, 28, 29
- labkey.domain.inferFields, 16, 18, 20–22, 24–26, 27, 29
- labkey.domain.save, 16, 18, 20–22, 24–26, 28, 28
- labkey.executeSql, 5, 13, 29, 41, 43, 46, 48, 49, 51, 53, 54, 74, 84, 93
- labkey.experiment.createData, 31, 33, 34, 37, 39
- labkey.experiment.createMaterial, 32, 32, 34, 37, 39
- labkey.experiment.createRun, 32, 33, 33, 37, 39
- labkey.experiment.SAMPLE\_DERIVATION\_PROTOCOL, 35
- labkey.experiment.saveBatch, 32–35, 36, 74
- labkey.experiment.saveRuns, 38
- labkey.getBaseUrl, 39
- labkey.getDefaultViewDetails, 40, 42, 43, 46, 48, 49, 51, 84
- labkey.getFolders, 41, 76–78, 80
- labkey.getLookupDetails, 41, 42, 42, 46, 48, 49, 51, 84
- labkey.getModuleProperty, 44
- labkey.getQueries, 41–43, 45, 48, 49, 51, 84
- labkey.getQueryDetails, 40–43, 46, 46, 49, 51, 84
- labkey.getQueryViews, 41–43, 46, 48, 48, 51, 84
- labkey.getRequestOptions, 50
- labkey.getSchemas, 41, 43, 46, 48, 49, 51, 84
- labkey.importRows, 5, 13, 30, 41, 43, 46, 48, 49, 51, 52, 54, 69, 84, 93
- labkey.insertRows, 5, 13, 30, 41, 43, 46, 48, 49, 51, 53, 53, 69, 84, 93
- labkey.makeRemotePath, 56
- labkey.pipeline.getFileStatus, 57, 59, 60, 62
- labkey.pipeline.getPipelineContainer, 58, 58, 60, 62
- labkey.pipeline.getProtocols, 58, 59, 59, 62
- labkey.pipeline.startAnalysis, 58–60, 61
- labkey.provenance.addRecordingStep, 13, 54, 62, 65–67, 93
- labkey.provenance.createProvenanceParams, 13, 54, 63, 63, 66, 67, 93
- labkey.provenance.startRecording, 13, 54, 63, 65, 65, 67, 93
- labkey.provenance.stopRecording, 13, 54, 63, 65, 66, 67, 93
- labkey.query.import, 53, 54, 68, 93
- labkey.rstudio.initReport, 70
- labkey.rstudio.initRStudio, 71
- labkey.rstudio.initSession, 71
- labkey.rstudio.isInitialized, 72
- labkey.rstudio.saveReport, 72
- labkey.saveBatch, 73
- labkey.security.createContainer, 42, 75,

- [77, 78, 80](#)
- labkey.security.deleteContainer, [42, 76, 76, 78, 80](#)
- labkey.security.getContainers, [42, 76, 77, 77, 80](#)
- labkey.security.impersonateUser, [78, 81, 105](#)
- labkey.security.moveContainer, [42, 76–78, 79](#)
- labkey.security.stopImpersonating, [79, 81, 105](#)
- labkey.selectRows, [5, 8, 13, 30, 41, 43, 46, 48, 49, 51, 53, 54, 74, 82, 93, 110](#)
- labkey.setCurlOptions, [85](#)
- labkey.setDebugMode, [86](#)
- labkey.setDefaults, [87](#)
- labkey.setModuleProperty, [88](#)
- labkey.transform.getRunPropertyValue, [89](#)
- labkey.transform.readRunPropertiesFile, [90](#)
- labkey.truncateTable, [91](#)
- labkey.updateRows, [5, 13, 30, 41, 43, 46, 48, 49, 51, 53, 54, 69, 74, 84, 92](#)
- labkey.webdav.delete, [94, 96, 97, 99–102, 104](#)
- labkey.webdav.downloadFolder, [95, 95, 97, 99–102, 104](#)
- labkey.webdav.get, [95, 96, 96, 99–102, 104](#)
- labkey.webdav.listdir, [95–97, 98, 100–102, 104](#)
- labkey.webdav.mkdir, [95–97, 99, 99, 101, 102, 104](#)
- labkey.webdav.mkDirs, [95–97, 99, 100, 100, 102, 104](#)
- labkey.webdav.pathExists, [95–97, 99–101, 102, 104](#)
- labkey.webdav.put, [95–97, 99–102, 103](#)
- labkey.whoAmI, [79, 81, 104](#)
- lsFolders, [5, 105, 107, 108](#)
- lsProjects, [106, 106, 108](#)
- lsSchemas, [106, 107, 107](#)
  
- makeFilter, [5, 13, 30, 41, 43, 46, 48, 49, 51, 53, 54, 74, 84, 93, 108](#)
  
- Rlabkey (Rlabkey-package), [3](#)
- Rlabkey-package, [3](#)
- RlabkeyUsersGuide, [110](#)
  
- saveResults, [8, 11, 111](#)