Package 'REDCapR'

September 5, 2014

Title Interaction between R and REDCap

Description Encapsulates functions to streamline calls from R

Version 0.4-11

Date 2014-09-05

Author Will Beasley, David Bard, Thomas Wilson

Maintainer 'Will Beasley' <wibeasley@hotmail.com>

URL http://ouhsc.edu/bbmc/

Depends R(>= 3.0.0),stats

Imports httr,plyr,stringr

Suggests devtools,knitr,methods,RODBC,testit,testthat

License GPL-3

LazyData TRUE

VignetteBuilder knitr

Roxygen list(wrap = TRUE)

R topics documented:

create_batch_glossary *Creates a* data.frame *that help batching long-running read and writes.*

Description

The function returns a data.frame that other functions use to separate long-running read and write REDCap calls into multiple, smaller REDCap calls. The goal is to (1) reduce the chance of time-outs, and (2) introduce little breaks between batches so that the server isn't continually tied up.

Usage

```
create_batch_glossary(row_count, batch_size)
```

Arguments

row_count	The number records in the large dataset, before it's split.
batch_size	The maximum number of subject records a single batch should contain.

Details

This function can also assist splitting and saving a large data.frame to disk as smaller files (such as a .csv). The padded columns allow the OS to sort the batches/files in sequential order.

Value

Currently, a data.frame is returned with the following columns,

1. id: an integer that uniquely identifies the batch, starting at 1.
2. start_index: the index of the first row in the batch. integer.
3. stop_index: the index of the last row in the batch. integer.
4. id_pretty: a character representation of id, but padded with zeros.
5. start_index: a character representation of start_index, but padded with zeros.
6. stop_index: a character representation of stop_index, but padded with zeros.
7. label: a character concatenation of id_pretty, start_index, and stop_index_pretty.

Author(s)

Will Beasley

See Also

See redcap_read for a function that uses create_batch_gloassary.

Examples

```
library(REDCapR) #Load the package into the current R session.
create_batch_glossary(100, 50)
create_batch_glossary(100, 25)
create_batch_glossary(100, 3)
d <- data.frame(
  recordid = 1:100,
  iv = sample(x=4, size=100, replace=TRUE),
  dv = rnorm(n=100)
)
create_batch_glossary(nrow(d), batch_size=40)
```

REDCapR *REDCapR*

Description

REDCapR

---

redcap_column_sanitize *Sanitize to adhere to REDCap character encoding requirements.*

Description

Replace non-ASCII characters with legal characters that won't cause problems when writing to a REDCap project.

Usage

```
redcap_column_sanitize(d, column_names = colnames(d),
  encoding_initial = "latin1", substitution_character = "?")
```

Arguments

d	The data.frame containing the dataset used to update the REDCap project. Required.
column_names	An array of character values indicating the names of the variables to sanitize. Optional.
encoding_initial	An array of character values indicating the names of the variables to sanitize. Optional.
substitution_character	The character value that replaces characters that were unable to be appropriatedly matched.

Details

Letters like an accented 'A' are replaced with a plain 'A'.

This is a thin wrapper around `base::iconv()`. The `ASCII//TRANSLIT` option does the actual transliteration work. As of `R 3.1.0`, the OSes use similar, but different, versions to convert the characters. Be aware of this in case you notice slight OS-dependent differences.

Value

A `data.frame` with same columns, but whose character values have been sanitized.

Author(s)

Will Beasley

Examples

```
# Examples are not shown because they require non-ASCII encoding,
#   which makes the package documentation less portable.
```

---

redcap_project *A* `Reference Class` *to make later calls to REDCap more convenient.*

Description

This `Reference Class` represents a REDCap project. Once some values are set that are specific to a REDCap project (such as the URI and token), later calls are less verbose (such as reading and writing data). The functionality

Fields

`redcap_uri` The URI (uniform resource identifier) of the REDCap project. Required.

`token` token The user-specific string that serves as the password for a project. Required.

Methods

`read(batch_size = 100L, interbatch_delay = 0, records = NULL, records_collapsed = "", fields = N`
Exports records from a REDCap project.

`write(ds_to_write, batch_size = 100L, interbatch_delay = 0, verbose = TRUE, cert_location = NULL`
Imports records to a REDCap project.

Examples

```
library(REDCapR) #Load the package into the current R session.
uri <- "https://bbmc.ouhsc.edu/redcap/api/"
token <- "D70F9ACD1EDD6F151C6EA78683944E98"
project <- redcap_project$new(redcap_uri=uri, token=token)
dsAll <- project$read()

#Demonstrate how repeated calls are more concise when the token and url aren't always passed.
dsThreeColumns <- project$read(fields=c("record_id", "sex", "height"))$data

idsOfMales <- dsThreeColumns[dsThreeColumns$sex==1, "record_id"]
```

```
idsOfShorties <- dsThreeColumns[dsThreeColumns$height < 40, "record_id"]

dsMales <- project$read(records=idsOfMales, batch_size=2)$data
dsShorties <- project$read(records=idsOfShorties)$data

## Not run:
#Switch the Genders
sex_original <- dsThreeColumns$sex
dsThreeColumns$sex <- (1 - dsThreeColumns$sex)
project$write(dsThreeColumns)

#Switch the Genders back
dsThreeColumns$sex <- sex_original
project$write(dsThreeColumns)

## End(Not run)
```

`redcap_read` *Read records from a REDCap project in subsets, and stacks them together before returning a* `data.frame`.

Description

From an external perspective, this function is similar to `redcap_read_oneshot`. The internals differ in that `redcap_read` retrieves subsets of the data, and then combines them before returning (among other objects) a single `data.frame`. This function can be more appropriate than `redcap_read_oneshot` when returning large datasets that could tie up the server.

Usage

```
redcap_read(batch_size = 100L, interbatch_delay = 0.5, redcap_uri, token,
  records = NULL, records_collapsed = "", fields = NULL,
  fields_collapsed = "", export_data_access_groups = FALSE,
  raw_or_label = "raw", verbose = TRUE, cert_location = NULL,
  id_position = 1L)
```

Arguments

`batch_size`	The maximum number of subject records a single batch should contain. The default is 100.
`interbatch_delay`	The number of seconds the function will wait before requesting a new subset from REDCap. The default is 0.5 seconds.
`redcap_uri`	The URI (uniform resource identifier) of the REDCap project. Required.
`token`	The user-specific string that serves as the password for a project. Required.
`records`	An array, where each element corresponds to the ID of a desired record. Optional.
`records_collapsed`	A single string, where the desired ID values are separated by commas. Optional.
`fields`	An array, where each element corresponds a desired project field. Optional.

`fields_collapsed`
: A single string, where the desired field names are separated by commas. Optional.

`export_data_access_groups`
: A boolean value that specifies whether or not to export the "redcap_data_access_group" field when data access groups are utilized in the project. Default is `FALSE`. See the details below.

`raw_or_label`
: A string (either `'raw'` or `'label'` that specifies whether to export the raw coded values or the labels for the options of multiple choice fields. Default is `'raw'`.

`verbose`
: A boolean value indicating if `messages` should be printed to the R console during the operation. Optional.

`cert_location`
: If present, this string should point to the location of the cert files required for SSL verification. If the value is missing or NULL, the server's identity will be verified using a recent CA bundle from the cURL website. See the details below. Optional.

`id_position`
: The column position of the variable that unique identifies the subject. This defaults to the first variable in the dataset.

Details

Specifically, it internally uses multiple calls to `redcap_read_oneshot` to select and return data. Initially, only primary key is queried through the REDCap API. The long list is then subsetted into partitions, whose sizes are determined by the `batch_size` parameter. REDCap is then queried for all variables of the subset's subjects. This is repeated for each subset, before returning a unified `data.frame`.

The function allows a delay between calls, which allows the server to attend to other users' requests.

Value

Currently, a list is returned with the following elements,

1. `data`: An R `data.frame` of the desired records and columns.
2. `success`: A boolean value indicating if the operation was apparently successful.
3. `status_codes`: A collection of http status codes, separated by semicolons. There is one code for each batch attempted.
4. `outcome_messages`: A collection of human readable strings indicating the operations' semicolons. There is one code for each batch attempted. In an unsuccessful operation, it should contain diagnostic information.
5. `records_collapsed`: The desired records IDs, collapsed into a single string, separated by commas.
6. `fields_collapsed`: The desired field names, collapsed into a single string, separated by commas.
7. `elapsed_seconds`: The duration of the function.

Author(s)

Will Beasley

References

The official documentation can be found on the REDCap wiki (https://iwg.devguard.com/trac/redcap/wiki/ApiDocumentation). Also see the 'API Examples' page on the REDCap wiki (https://iwg.devguard.com/trac/redcap/wiki/ApiExamples). A user account is required to access the wiki, which typically is granted only to REDCap administrators. If you do not

The official cURL site discusses the process of using SSL to verify the server being connected to.

Examples

```
## Not run:
library(REDCapR) #Load the package into the current R session.
uri <- "https://bbmc.ouhsc.edu/redcap/api/"
token <- "9A81268476645C4E5F03428B8AC3AA7B"
redcap_read(batch_size=2, redcap_uri=uri, token=token)

## End(Not run)
```

`redcap_read_oneshot` *Read/Export records from a REDCap project.*

Description

This function uses REDCap's API to select and return data.

Usage

```
redcap_read_oneshot(redcap_uri, token, records = NULL,
  records_collapsed = "", fields = NULL, fields_collapsed = "",
  export_data_access_groups = FALSE, raw_or_label = "raw", verbose = TRUE,
  cert_location = NULL)
```

Arguments

`redcap_uri`	The URI (uniform resource identifier) of the REDCap project. Required.
`token`	The user-specific string that serves as the password for a project. Required.
`records`	An array, where each element corresponds to the ID of a desired record. Optional.
`records_collapsed`	A single string, where the desired ID values are separated by commas. Optional.
`fields`	An array, where each element corresponds a desired project field. Optional.
`fields_collapsed`	A single string, where the desired field names are separated by commas. Optional.
`export_data_access_groups`	A boolean value that specifies whether or not to export the "redcap_data_access_group" field when data access groups are utilized in the project. Default is `FALSE`. See the details below.
`raw_or_label`	A string (either `'raw'` or `'label'` that specifies whether to export the raw coded values or the labels for the options of multiple choice fields. Default is `'raw'`.

verbose	A boolean value indicating if messages should be printed to the R console during the operation. Optional.
cert_location	If present, this string should point to the location of the cert files required for SSL verification. If the value is missing or NULL, the server's identity will be verified using a recent CA bundle from the cURL website. See the details below. Optional.

Details

I like how PyCap creates a 'project' object with methods that read and write from REDCap. However this isn't a style that R clients typically use. I like the logic that it's associated with a particular REDCap project that shouldn't change between calls. As a compromise, I think I'll wrap the uri, token, and cert location into a single S4 object that's passed to these methods. It will make these calls take less space.

The 'REDCapR' package includes a recent version of the Bundle of CA Root Certificates from the official cURL site. This version is used by default, unless the 'cert_location' parameter is given another location.

If you do not pass in this export_data_access_groups value, it will default to FALSE. The following is from the API help page for version 5.2.3: This flag is only viable if the user whose token is being used to make the API request is *not* in a data access group. If the user is in a group, then this flag will revert to its default value.

Value

Currently, a list is returned with the following elements,

1. data: An R data.frame of the desired records and columns.
2. success: A boolean value indicating if the operation was apparently successful.
3. status_code: The http status code of the operation.
4. outcome_message: A human readable string indicating the operation's outcome.
5. records_collapsed: The desired records IDs, collapsed into a single string, separated by commas.
6. fields_collapsed: The desired field names, collapsed into a single string, separated by commas.
7. elapsed_seconds: The duration of the function.
8. raw_text: If an operation is NOT successful, the text returned by REDCap. If an operation is successful, the 'raw_text' is returned as an empty string to save RAM.

Author(s)

Will Beasley

References

The official documentation can be found on the 'API Examples' page on the REDCap wiki (https://iwg.devguard.com/trac/redcap/wiki/ApiExamples). A user account is required.

The official cURL site discusses the process of using SSL to verify the server being connected to.

Examples

```
## Not run:
library(REDCapR) #Load the package into the current R session.
uri <- "https://bbmc.ouhsc.edu/redcap/api/"
token <- "9A81268476645C4E5F03428B8AC3AA7B"
#Return all records and all variables.
ds_all_rows_all_fields <- redcap_read_oneshot(redcap_uri=uri, token=token)$data

#Return only records with IDs of 1 and 3
desired_records_v1 <- c(1, 3)
ds_some_rows_v1 <- redcap_read_oneshot(
   redcap_uri=uri,
   token=token,
   records=desired_records_v1
)$data

#Return only the fields recordid, first_name, and age
desired_fields_v1 <- c("recordid", "first_name", "age")
ds_some_fields_v1 <- redcap_read_oneshot(
   redcap_uri=uri,
   token=token,
   fields=desired_fields_v1
)$data

## End(Not run)
```

`redcap_write` *Write/Import records to a REDCap project.*

Description

This function uses REDCap's API to select and return data.

Usage

```
redcap_write(ds_to_write, batch_size = 100L, interbatch_delay = 0.5,
  redcap_uri, token, verbose = TRUE, cert_location = NULL)
```

Arguments

`ds_to_write`	The `data.frame` to be imported into the REDCap project. Required.
`batch_size`	The maximum number of subject records a single batch should contain. The default is 100.
`interbatch_delay`	The number of seconds the function will wait before requesting a new subset from REDCap. The default is 0.5 seconds.
`redcap_uri`	The URI (uniform resource identifier) of the REDCap project. Required.
`token`	The user-specific string that serves as the password for a project. Required.
`verbose`	A boolean value indicating if `messages` should be printed to the R console during the operation. Optional.

`cert_location` If present, this string should point to the location of the cert files required for SSL verification. If the value is missing or NULL, the server's identity will be verified using a recent CA bundle from the cURL website. See the details below. Optional.

Details

The 'REDCapR' package includes a recent version of the Bundle of CA Root Certificates from the official cURL site. This version is used by default, unless the 'cert_location' parameter is given another location.

Currently, the function doesn't modify any variable types to conform to REDCap's supported variables. See `validate_for_write` for a helper function that checks for some common important conflicts.

Value

Currently, a list is returned with the following elements,

1. `success`: A boolean value indicating if the operation was apparently successful.
2. `status_code`: The http status code of the operation.
3. `outcome_message`: A human readable string indicating the operation's outcome.
4. `records_affected_count`: The number of records inserted or updated.
5. `affected_ids`: The subject IDs of the inserted or updated records.
6. `elapsed_seconds`: The duration of the function.

Author(s)

Will Beasley

References

The official documentation can be found on the 'API Examples' page on the REDCap wiki (`https://iwg.devguard.com/trac/redcap/wiki/ApiExamples`). A user account is required.

The official cURL site discusses the process of using SSL to verify the server being connected to.

`redcap_write_oneshot` *Write/Import records to a REDCap project.*

Description

This function uses REDCap's API to select and return data.

Usage

```
redcap_write_oneshot(ds, redcap_uri, token, verbose = TRUE,
  cert_location = NULL)
```

Arguments

ds	The data.frame to be imported into the REDCap project. Required.
redcap_uri	The URI (uniform resource identifier) of the REDCap project. Required.
token	The user-specific string that serves as the password for a project. Required.
verbose	A boolean value indicating if messages should be printed to the R console during the operation. Optional.
cert_location	If present, this string should point to the location of the cert files required for SSL verification. If the value is missing or NULL, the server's identity will be verified using a recent CA bundle from the cURL website. See the details below. Optional.

Details

The 'REDCapR' package includes a recent version of the Bundle of CA Root Certificates from the official cURL site. This version is used by default, unless the 'cert_location' parameter is given another location.

Currently, the function doesn't modify any variable types to conform to REDCap's supported variables. See validate_for_write for a helper function that checks for some common important conflicts.

Value

Currently, a list is returned with the following elements,

1. success: A boolean value indicating if the operation was apparently successful.
2. status_code: The http status code of the operation.
3. outcome_message: A human readable string indicating the operation's outcome.
4. records_affected_count: The number of records inserted or updated.
5. affected_ids: The subject IDs of the inserted or updated records.
6. elapsed_seconds: The duration of the function.
7. raw_text: If an operation is NOT successful, the text returned by REDCap. If an operation is successful, the 'raw_text' is returned as an empty string to save RAM.

Author(s)

Will Beasley

References

The official documentation can be found on the 'API Examples' page on the REDCap wiki (https://iwg.devguard.com/trac/redcap/wiki/ApiExamples). A user account is required.

The official cURL site discusses the process of using SSL to verify the server being connected to.

Examples

```
## Not run:
#Define some constants
uri  <- "https://bbmc.ouhsc.edu/redcap/api/"
token <- "D70F9ACD1EDD6F151C6EA78683944E98"
```

```
# Read the dataset for the first time.
result_read1 <- redcap_read_oneshot(redcap_uri=uri, token=token)
ds1 <- result_read1$data
ds1$telephone
# The line above returns something like this (depending on its previous state).
# [1] "(432) 456-4848" "(234) 234-2343" "(433) 435-9865" "(987) 654-3210" "(333) 333-4444"

# Manipulate a field in the dataset in a VALID way
ds1$telephone <- sprintf("(405) 321-%1$i%1$i%1$i%1$i", seq_len(nrow(ds1)))

ds1 <- ds1[1:3, ]
ds1$age <- NULL; ds1$bmi <- NULL #Drop the calculated fields before writing.
result_write <- REDCapR::redcap_write_oneshot(ds=ds1, redcap_uri=uri, token=token)

# Read the dataset for the second time.
result_read2 <- redcap_read_oneshot(redcap_uri=uri, token=token)
ds2 <- result_read2$data
ds2$telephone
# The line above returns something like this.  Notice only the first three lines changed.
# [1] "(405) 321-1111" "(405) 321-2222" "(405) 321-3333" "(987) 654-3210" "(333) 333-4444"

# Manipulate a field in the dataset in an INVALID way.  A US exchange can't be '111'.
ds1$telephone <- sprintf("(405) 111-%1$i%1$i%1$i%1$i", seq_len(nrow(ds1)))

# This next line will throw an error.
result_write <- REDCapR::redcap_write_oneshot(ds=ds1, redcap_uri=uri, token=token)
result_write$raw_text

## End(Not run)
```

`retrieve_token` *Read a token from a (non-REDCap) database.*

Description

These functions are not essential to calling the REDCap API, but instead are functions that help manage tokens securely.

Usage

```
retrieve_token_mssql(dsn, project_name, channel = NULL,
  schema_name = "[Redcap]", procedure_name = "[prcToken]",
  variable_name_project = "@RedcapProjectName", field_name_token = "Token")
```

Arguments

`dsn`	A DSN on the local machine that points to the desired MSSQL database. Required.
`project_name`	The friendly/shortened name given to the REDCap project in the MSSQL table. Notice this isn't necessarily the same name used by REDCap. Required
`channel`	An *optional* connection handle as returned by `RODBC::odbcConnect`. See Details below. Optional.

`schema_name` The schema used within the database. Note that MSSQL uses the more conventional definition of schema than MySQL. Defaults to `'[Redcap]'`. Optional.

`procedure_name` The stored procedure called to retrieve the token. Defaults to `'[prcToken]'`. Optional.

`variable_name_project`
The variable declared within the stored procedure that contains the desired project name. Optional.

`field_name_token`
The field/column/variable name in the database table containing the token values. Defaults to `'Token'`. Optional.

Details

If no `channel` is passed, one will be created at the beginning of the function, and destroyed at the end. However if a channel is created, it's the caller's responsibility to destroy this resource. If you're making successive calls to the database, it might be quicker to create a single `channel` object and batch the calls together. Otherwise, the performance should be equivalent.

If you create the `channel` object yourself, consider wrapping calls in a `base::tryCatch` block, and closing the channel in its `finally` expression; this helps ensure the expensive database resource isn't held open unnecessarily. See the internals of `retrieve_token_mssql` for an example of closing the `channel` in a `tryCatch` block.

If the database elements are create with the script provided in security_database, the default values will work.

Value

The token, which is a 32 character string.

Note

We use Microsoft SQL Server, because that fits our University's infrastructure the easiest. But this approach theoretically can work with any LDAP-enabled database server. Please contact us if your institution is using something other than SQL Server, and would like help adapting this approach to your infrastructure.

Author(s)

Will Beasley

Examples

```
## Not run:
library(REDCapR) #Load the package into the current R session.

##
## Rely on `retrieve_token()` to create & destory the channel.
##
dsn <- "TokenSecurity"
project <- "DiabetesSurveyProject"
token <- retrieve_token(dsn=dsn, project_name=project)

##
## Create & close the channel yourself, to optimize repeated calls.
```

```
##
dsn <- "TokenSecurity"
project1 <- "DiabetesSurveyProject1"
project2 <- "DiabetesSurveyProject2"
project3 <- "DiabetesSurveyProject3"

channel <- RODBC::odbcConnect(dsn=dsn)
token1 <- retrieve_token(dsn=dsn, project_name=project1)
token2 <- retrieve_token(dsn=dsn, project_name=project2)
token3 <- retrieve_token(dsn=dsn, project_name=project3)
RODBC::odbcClose(channel)

## End(Not run)
```

`security_database` *Create an auxiliary database that handles sensitive values.*

Description

The SQL code below adds schemas, a table and two stored procedures to an existing Microsoft SQL Database. This second database is not essential to calling the REDCap API, but it helps manage tokens securely.

Details

This database contains the tokens and other sensitive content (such as passwords, API tokens, and file paths) that should not be stored in a Git repository (even a private Git repository). These passwords can be retrieved by `retrieve_token_mssql`.

After executing the SQL code in an existing database, create an ODBC DSN on *each* client machine that calls the database. Download the most recent drivers (as of Sept 2014, the most recent version is 11 for Windows and Linux.), then run the wizard. Many values in the wizard will remain at the default values. Here are the important ones to change.

1. Set the DSN's `"name"` field to whatever is used in the repository's R code.
2. Set the authenticity method to `Integrated Windows authentication`.
3. Set the `default database` to the name of the database that containing the tokens (*i.e.*, corresponding to the SQL code below in the example).

Note

We use Microsoft SQL Server, because that fits our University's infrastructure the easiest. But this approach theoretically can work with any LDAP-enabled database server. Please contact us if your institution is using something other than SQL Server, and would like help adapting this approach to your infrastructure.

Author(s)

Will Beasley

Examples

```
## Not run:
------- SQL code to create necessary components in a Microsoft SQL Sever database -------
--
-- Create two schemas.
-- The first scehma is accessible by all REDCap API users.
-- The second scehma is restricted to administrators.
--
CREATE SCHEMA [Redcap]
CREATE SCHEMA [RedcapPrivate]
GO

--
-- Create a table to contain the token
--
CREATE TABLE [RedcapPrivate].[tblToken](
  [ID] [smallint] IDENTITY(1,1) NOT NULL,
  [Username] [varchar](30) NOT NULL,
  [RedcapProjectName] [varchar](90) NOT NULL,
  [RedcapProjectID] [smallint] NOT NULL,
  [Token] [char](32) NOT NULL,
 CONSTRAINT [PK_RedcapApiTokens] PRIMARY KEY CLUSTERED(
  [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
  ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

CREATE NONCLUSTERED INDEX [IX_tblToken_UniqueUsernameProjectID] ON [RedcapPrivate].[tblToken](
  [Username] ASC,
  [RedcapProjectID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
  ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

CREATE NONCLUSTERED INDEX [IX_tblToken_UniqueUsernameProjectName] ON [RedcapPrivate].[tblToken](
  [Username] ASC,
  [RedcapProjectName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF,
  ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO

--
-- Create a stored procedure for users to call to retrieve the token.
-- Notice it should a different (and more permissive) schema than the table.
--
CREATE PROCEDURE [Redcap].[prcToken]
  @RedcapProjectName varchar(30) -- Add the parameters for the stored procedure here
AS
BEGIN
  -- SET NOCOUNT ON added to prevent extra result sets from interfering with SELECT statements.
  SET NOCOUNT ON;

  SELECT Token FROM [RedcapPrivate].[tblToken]
  WHERE Username=system_user AND RedcapProjectName=@RedcapProjectName
END
```

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

`validate_for_write` *Inspect a* `data.frame` *to anticipate problems before writing to a REDCap project.*

Description

This set of functions inspect a `data.frame` to anticipate problems before writing with REDCap's API.

Usage

```
validate_for_write( d )

validate_no_logical( d )

validate_no_uppercase( d )
```

Arguments

`d`	The `data.frame` containing the dataset used to update the REDCap project. Required.

Details

All functions listed in the Usage section above inspect a specific aspect of the dataset. The `validate_for_read()` function executes all these individual validation checks. It allows the client to check everything with one call.

Value

A `data.frame`, where each potential violation is a row. The two columns are:

1. `field_name`: The name of the `data.frame` that might cause problems during the upload.
2. `field_index`: The position of the field. (For example, a value of '1' indicates the first column, while a '3' indicates the third column.)
3. `concern`: A description of the problem potentially caused by the `field`.
4. `suggestion`: A *potential* solution to the concern.

Author(s)

Will Beasley

Examples

```
d <- data.frame(
  recordid = 1:4,
  flag_logical = c(TRUE, TRUE, FALSE, TRUE),
  flag_Uppercase = c(4, 6, 8, 2)
)
validate_for_write(d = d)
```

Index