Package: prolific.api (via r-universe)

September 4, 2024

Title A User-Friendly Interface for Accessing the Prolific API

Version 0.5.1

Description A user-friendly interface for creating and managing empirical crowd-sourcing studies via API access to <https://www.prolific.co>.

License GPL (>= 3)

Imports data.table (>= 1.14.6), jsonlite (>= 1.8.4), methods, utils

SystemRequirements curl (https://curl.se/)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.1

Suggests htmltools, knitr, reactable, rmarkdown

VignetteBuilder knitr

Repository https://simon-lenau.r-universe.dev

RemoteUrl https://github.com/simon-lenau/prolific.api

RemoteRef HEAD

RemoteSha b83de33f8bbd971d9ef86239fec878749deaf788

Contents

prolific.api-package *R interface to the Prolific API*

Description

A set of user-friendly functionalities for creating and managing potentially large numbers of studies on the Prolific platform via its API. The platform is designed for recruiting participants for empirical studies via crowd-sourcing, allowing to apply a number of prescreening characteristics to target specific groups of participants for a study.

Object classes

`prolific.api` provides three `ReferenceClasses` to access the Prolific API, namely `api_access`, `prolific_study` and `prolific_prescreener`. An overview is provided below.

api_access:

`api_access` objects provide functionalities for accessing the API, which requires to specify a valid API token.

prolific_study:

`prolific_study` objects represent studies to be created or managed on Prolific. Users can create new studies, or retrieve existing studies from Prolific and apply updates to them.

prolific_prescreener:

`prolific_prescreener` objects characterize the participants to be selected for a certain `prolific_study`, i.e. the requirements that a person needs to meet to be recruited for the `study`.

Authentication

A researcher account on `Prolific` is required to use the functionalities of this package. To use this account, a valid Prolific API token must be specified for authentication. These tokens are *workspace-specific* and can be managed in the `Settings -> Go to API token` page menu (https://app.prolific.co/researcher/workspaces/workspace_id/settings/tokens for an existing `workspace_id`).

api_access *Prolific API access*

Description

This class provides functionalities for accessing the Prolific API. The core method for this purpose is `access`, which can be used to create, review, change, manage and delete studies on the Prolific platform.
The fields and methods are available as in RefClass or S4 objects (see examples).

Fields

`accessors` (`character`):
The commands for accessing the API. The command for each type of access method can be altered using this field. The default is

```
accessors = c(
   get    = "curl",
   post   = "curl -X POST",
   put    = "curl -X PUT",
   patch  = "curl -X PATCH",
   delete = "curl -X DELETE"
)
```

Note: *A value for each of the names* (`get`, `post`, `put`, `patch` and `delete`) *is required, as these represent the methods that can be used when accessing the API.*

`api_token` (`character`):
The Prolific API token.

`entrypoint` (`character`):
The API's entrypoint URL.

Methods

`access`**:**

Main method for accessing the Prolific API

Parameters*:*

`endpoint` (`character`):
The endpoint to access. If this is a vector, its elements are collapsed by `'/'`.

`method` (`character`):
The method to use. One of `get`, `post`, `place`, `patch` and `delete`. The commands associated with each method are defined in the `accessors` field of the `api_access` object.

`data` (`json string`, `json file`, `list`, `prolific_study object` or `NULL`)
The data to be transfered in the body of the API call. R-objects are converted to a `json string` using `jsonlite:toJSON`. `NULL` means that no data is transfered.

`as_list` (`logical`):
Whether the return of the API call should be converted to a list or (if applicable) prolific_study object, rather than returned as the raw `json string`.

Return Value*:*
A `list` or `json string`, depending on argument `as_list`.

Usage*:*

```
api_access$access(
     endpoint,
     method,
     data,
     as_list
)
```

`check_authorization`**:**

Check whether the API authorization works

Return Value*:*

A `logical` value that indicates whether the API authorization works.

Usage*:*

```
api_access$check_authorization()
```

Examples

```
library(prolific.api)

# Create API access
prolific_api_access <- api_access(api_token = "<api_token>")

# View fields
## RefClass Methods
prolific_api_access$accessors
prolific_api_access$api_token
prolific_api_access$entrypoint

## S4 Methods
accessors(prolific_api_access)
api_token(prolific_api_access)
entrypoint(prolific_api_access)

# Change fields
# (this is usually only required for the api_token)
# replace <new_token> in the by the actual API token
# before running these lines
## Not run:
## RefClass Method
prolific_api_access$api_token <- "<new_token>"
## S4 Method
api_token(prolific_api_access) <- "<new_token>"

## End(Not run)


# Note: For the following code to work,
# you have to replace <new_token> in the lines above by the actual API token
## Not run:
# Check wheter Authorization is working
## RefClass Method
prolific_api_access$check_authorization()
## S4 Method
check_authorization(prolific_api_access)

# Obtain list of existing studies
## RefClass Method
list_of_studies <-
    prolific_api_access$access(
```

```
        endpoint = "studies",
        method = "get",
        as_list = TRUE
    )
## S4 Method
list_of_studies2 <-
    access(
        prolific_api_access,
        endpoint = "studies",
        method = "get",
        as_list = TRUE
    )

## End(Not run)
```

`prolific_prescreener` *Prolific prescreening requirement*

Description

Class that represents prescreening requirements to characterize the participants to be selected for a certain `study on Prolific`, i.e. the persons to be recruited via Prolific. `prolific_prescreener objects` are therefore mainly used in the `eligibility_requirements` field of `prolific_studys`. *The fields and methods are available as in RefClass or S4 objects (see examples).*

The section *'Setting up prescreeners for Prolific'* below provides an overview and examples of how to specify prescreening requirements.

Fields

`title` (`character`):

A *valid* title for a single prescreener that is available on the Prolific platform. To be valid, this title *must* appear in the list of prescreeners obtainable from the Prolific API.
See the section *'Setting up prescreeners for Prolific'* as well as the prolific.api package vignette.

`constraints` (`list`):

The *valid* constraints for this particular prescreener.
When creating a `prolific_prescreener object`, an arbitrary number of constraints can be specified using `named or unnamed custom arguments`. In the **named** case,

```
name_1 = value_1,...,name_i = value_i,
```

`name = value` pairs are used to set the constraints and values. Using the **unnamed** case

```
name_1,...,name_i
```

allows to ommit the values for prescreeners where `value_1 = ... = value_i = TRUE`. In that way, users can simply provide the names of the groups to be recruited. See the section *'Setting up prescreeners for Prolific'* as well as the *examples* and prolific.api package vignette.

Setting up prescreeners for Prolific

Prescreeners are used to select participants for a `prolific_study` that meet certain characteristics. In most cases, this selection is done with regard to the answers the participants gave in a survey conducted by Prolific across all its members.

Choosing a prescreening variable:

At the moment, there are `265` variables which can be used to recruit specific subgroups from Prolific. To obtain a list of all available prescreening variables, use

```
table_of_prescreeners <-
          prescreeners(prolific_api_access)
```

where `prolific_api_access` is an `api_access object` with a valid `api_token`.

A prescreening variable is determined by the `title` field of the `prolific_prescreener object`. To be valid, this `title` **must** appear in the `title` column of the resulting `table_of_prescreeners`.

Setting constraints for a particular prescreening variable:

The constraints are specified in the form

```
name_1 = value_1,
...,
name_n = value_n
```

or

```
name_1,
...,
name_n
```

For most prescreeners, the values `value_1` ... `value_n` are `logical` values to select participants that gave a certain answer in some pre-screening question. In this case, specifying

```
name_i = TRUE
```

for the prescreener means that participants who gave answer `name_i` are eligible for the study. However, keep in mind there are some prescreeners that work in the opposite way, e.g. to specify a list of participants to be exluded (see the sections *'Ex- or include a list of specific participants'* and *'Ex- or include all participants from previous studies'* below).

For all cases where the values `value_1` ... `value_n` are `logical`,

```
name_1,
...,
name_n
```

is an equivalent shortcut for

```
name_1 = TRUE,
...,
name_n = TRUE
```

.

Yet, the constraint values are not always of type `logical`. In particular, there are prescreeners that allow to select participants lying within a certain range of a `numerical variable`. For example,

this is the case when selecting participants who are in a certain age bracket, where lower and upper boundary for a person's age are specified in the constraints. In this case, `value_1`, ..., `value_n` in the above specification need to be numeric as well, and **must be named** e.g. as in

```
min_age = 50,
max_age = 60
```

for selecting participants between age 50 and 60 for the study.

The names `name_1`, ..., `name_n` are always taken literally. This means that they are not automatically evaluated. Enclosing a name in an `eval()` command forces it to be evaluated rather than taken literally. This is important for example in cases where the categories are stored in a list (see the section *'Examples for prolific_prescreeners'* for an example).

To obtain the list of possible constraints for a particular prescreener with a *valid* title `"the_title"` as described above, use

```
table_of_constraints <-
          prescreeners(prolific_api_access,
                       filter=expression(title==c("the_title")),
                       show_full=TRUE)
```

The names `name_1`, ..., `name_n` of the `constraints` list should come from a single (typically the *name*) column of the resulting `table_of_constraints`, the respective list elements represent the values that participants have to meet.

To make this a bit clearer, the following section provides examples for setting up prescreening requirements.

Examples for `prolific_prescreeners`:

Nationality requirements For example, a study can be set to exclusively target participants who currently live in the UK or the USA by using

```
residential_prescreener <- prolific_prescreener(
    title = "Current Country of Residence",
    "United Kingdom", "United States"
 )
```

or equivalently

```
list_of_countries <- list(
    country_1="United Kingdom",
    country_2="United States")

residential_prescreener <- prolific_prescreener(
    title = "Current Country of Residence",
    eval(list_of_countries$country_1),
    eval(list_of_countries$country_2)
)
```

Note that `"Current Country of Residence"` appears in the *title* column of `table_of_prescreeners`, and `"United Kingdom"` as well as `"United States"` appear in the *name* column of the resulting `table_of_constraints` described in the previous sections. Furthermore, note the use of `eval()` to force evaluation of `list_of_countries$country_1` and `list_of_countries$country_2`.

Age requirements Similarly, selecting participants who fall in the age range between 50 and 60 can be achieved through

```
age_prescreener <- prolific_prescreener(
    title = "Age",
    "min_age" = 50,
    "max_age" = 60
)
```

Ex- or include a list of specific participants Specific participants can be in- or excluded from a study, for example if they participated in previous studies. This can be done in form of black- or whitelists.
Consider two fictional participants with Prolific id's `111` and `222`. These can be specifically excluded by using the exclusion list defined by

```
exclude_list_participants <- prolific_prescreener(
    title = "Custom Blacklist",
    "111","222"
)
```

To exclusively recruit exactly these two participanty, use the include list defined by

```
include_list_participants <- prolific_prescreener(
    title = "Custom Whitelist",
    "111","222"
)
```

Note: The IDs for these constraints need to be valid Prolific IDs when creating a study. The above example for fictional IDs `111` and `222` will therefore always fail.

Ex- or include all participants from previous studies You can not only blacklist single participants, but also the group(s) of participants who participated in of one or multiple of your previous studies.
To exclude all participants from two fictional studies with IDs `ABC` and `DEF`, specify the prescreener

```
exclude_list_studies <- prolific_prescreener(
    title = "Exclude participants from previous studies",
    "ABC","DEF"
)
```

To exclusively recruit participants from these studies, use

```
include_list_studies <- prolific_prescreener(
    title = "Include participants from previous studies",
    "ABC","DEF"
)
```

Note: The IDs for these constraints need to be valid Study IDs when creating a study. The above example for fictional IDs `ABC` and `DEF` will therefore always fail.

Methods

`validity_check`**:**

Check whether the prescreener is valid in terms of the Prolific API.
Note: For checking a prescreener's validity, an `api_access object` that passes `check_authorization()` needs to be available. It suffices if any such `api_access object` is specified, since the reference to it is determined automatically.

Return Value*:*

- If the prescreener is valid: A `logical` value indicating that the study is valid
- If the prescreener is not valid: A `character` vector that lists the prescreener's issues.

Usage:

```
prescreener$validity_check()
```

Examples

```
library("prolific.api")

prolific_api_access <- api_access(api_token = "<api_token>")

# Create a new study with two of the prescreening constraints
#    from the help section 'Examples for prolific_prescreeners'
#    in this package's documentation.
fancy_new_study_with_prescreeners <- prolific_study(
    name = "A fancy study on Prolific",
    description = "Fancy description",
    external_study_url = "https://www.my_fancy_study_url.com",
    completion_code = "123ab456cd78",
    estimated_completion_time = 1,
    reward = 1,
    total_available_places = 1,
    eligibility_requirements = list(
        # Include only persons who live in the UK or the US
        prolific_prescreener(
            title = "Current Country of Residence",
            "United Kingdom", "United States"
        ),
        # Include participants only if they are between
        #    50 and 60 years old
        prolific_prescreener(
            title = "Age",
            "min_age" = 50,
            "max_age" = 60
        )
    )
)

# Note: For the following code to work,
# you have to replace <api_token> in the code above by the actual API token
## Not run:
# Post the 'fancy_new_study_with_prescreeners' to Prolific,
#    i.e. create it as a draft study on the platform
prolific_api_access$access(
    endpoint = "studies",
    method = "post",
    data = fancy_new_study_with_prescreeners
)

# Success: fancy_new_study_with_prescreeners got an ID - it is now a draft study on Prolific!
# You can also inspect the study and requirements in the Prolific Web UI now.
fancy_new_study_with_prescreeners$id
```

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

`prolific_study` *Prolific study*

Description

Class that represents Prolific studies, such that they can be transferred to or from the Prolific API. This allows to create, review and update studies.
The fields and methods are available as in RefClass or S4 objects (see examples and the prolific.api package vignette).

API access to interact with the Prolific platform is done by using objects from the `api_access class`, i.e. `prolific_studies` are intended to be transferred as bodies in calls to the Prolific API (see examples).

Fields

`id` (`character`):
The study's ID on Prolific.
Note: This ID is set by Prolific and can not be changed by the user
(see the '*Further (read-only) fields*' section below).

`name` (`character`):
Public name or title of the study *(will be publicly visible when publishing the study).*

`internal_name` (`character`):
Internal name of the study *(not shown to participants).*

`description` (`character`):
Description of the study *(will be publicly visible when publishing the study).*

`external_study_url` (`character`):
URL of the survey or experiment the participants will be redirected to *(will be publicly visible when publishing the study).*
Note:

- The URL must be valid at the time the study is created on the Prolific platform.
- For the use of URL parameters, see field `url_parameters`.

`url_parameters` (`list`):
A named list of URL parameters that is appended to `external_study_url`. The default

```
list(
     prolific_id = "{%PROLIFIC_PID%}",
     study_id = "{%STUDY_ID%}",
     session_id = "{%SESSION_ID%}"
)
```

is used for passing the participant's, study's and session's ID from Prolific to the data collection website.

`prolific_id_option` (`character`):
: This determines the method of passing the respondent's Prolific ID.
Valid options are:
- `"url_parameters"` for passing the ID as URL parameter `{%PROLIFIC_PID%}`
- `"question"` for letting the respondents enter their ID (e.g. via copy & paste), or
- `"not_required"` if the Prolific ID is not to be passed.

`completion_code` (`character`):
: The completion code that is provided to participants after completing the study. This code is used to prove that a participant completed the study. It is therefore *visible for participants after completing the study.*

`completion_option` (`character`):
: This determines the method for passing the `completion_code`.
Valid options are:
- `"url"`for passing the code as URL parameter when redirecting participants back to Prolific after completing the study, or
- `"code"`for providing a code for copy and paste.

`total_available_places` (`integer`):
: The number of participant you would like to recruit in the study *(will be publicly visible when publishing the study).*

`estimated_completion_time` (`integer`):
: The estimated time it takes to complete the study, *in minutes (will be publicly visible when publishing the study).*

`maximum_allowed_time` (`integer`):
: The maximum allowed time for participants to complete the study, *in minutes*.

`reward` (`integer`):
: The amount of money (in pence) you pay for completing the study *(will be publicly visible when publishing the study).*
Note: Compensation...

`eligibility_requirements` (`list`):
: A list containing `prolific_prescreener objects` that characterize the participants to be recruited. **Note:**
- `NULL` means that every participant can see and complete the study.
- **Only persons fulfiling these requirements will be able to participate in the study.**

`device_compatibility` (`character`):
: **Note:** `NULL` means that all options are available.

`peripheral_requirements` (`character`):
: A vector of technical requirements that participants have to fulfill to complete the study. One or multiple values from

```
c("audio", "camera", "download", "microphone")
```

Note: `NULL` means that none of the requirements is needed.

`naivety_distribution_rate` (`numeric`):
: A value between `1` and `0` that controls the balance between speed of your study and the naivety of the participants.
Prolific's description of this field is rather vague, but it seems to imply that

- `1` means that less trained or 'professional' participants will have access to the study.
- `0` means that all eligible participants will have access to the study at the same time.
- `values between 0 and 1` represent a tradeoff between both options.

`further_fields` (`list`):
Prolific studies can have various further fields, which (if used) are stored in `further_fields`. These fields are read-only, and determined by Prolific. See the '*Further (read-only) fields*' section below for a list of these read-only fields.

`...` (further arguments):
Will be added to the `further_fields` field of the `prolific_study` (see above).

Types of fields

Required fields are required for creating a study on Prolific.
The values for all of these except `completion_option` and `prolific_id_option` should be specified before publishing a study. Default values are only placeholders.

Optional fields are writable, but optional for Prolific.
The user can but does not have to set these fields when creating a study.
The required and optional fields are:

Required fields	**Optional fields**
`completion_code`	`device_compatibility`
`completion_option`	`internal_name`
`description`	`maximum_allowed_time`
`eligibility_requirements`	`naivety_distribution_rate`
`estimated_completion_time`	`peripheral_requirements`
`external_study_url`	`url_parameters`
`name`	
`prolific_id_option`	
`reward`	
`total_available_places`	

Further (read-only) fields contain information that is determined internally by Prolific and read-only.
The `id`-field is of particular relevance. Once a study is created via API access, it is **obtained from the API and stored in the `prolific_study` object**, since it can be used to update, manage or delete a study.
To fully represent the information that is obtainable from the Prolific API, the `further_fields` list can contain some or all of the entries listed below. The corresponding overview provided in the Prolific API documentation currently seems to be work in progress.

`_links`	`average_reward_per_hour`
`average_reward_per_hour_without_adjustment`	`average_time_taken`
`currency_code`	`date_created`
`device_compatibility`	`discount_from_coupons`
`eligible_participant_count`	`estimated_reward_per_hour`

fees_per_submission	fees_percentage
has_had_adjustment	internal_name
is_pilot	is_underpaying
last_email_update_sent_datetime	maximum_allowed_time
minimum_reward_per_hour	naivety_distribution_rate
number_of_submissions	peripheral_requirements
pilot_test_steps_state	places_taken
project	publish_at
published_at	publisher
quota_requirements	receipt
representative_sample	representative_sample_fee
researcher	reward_level
share_id	stars_remaining
status	study_type
total_cost	total_participant_pool
vat_percentage	workspace

Methods

`validity_check`**:**

Check whether the study is valid in terms of the Prolific API.
Note: For checking the validity of the `eligibility_requirements`, an `api_access object` that passes `check_authorization()` needs to be available. It suffices if any such `api_access object` is specified, since the reference to it is determined automatically.

Return Value*:*

- If the study is valid: A `logical` value indicating that the study is valid
- If the study is not valid: A `character` vector that lists the studie's issues.

Usage*:*

```
prolific_study$validity_check()
```

Examples

```
library(prolific.api)

prolific_api_access <- api_access(api_token = "<api_token>")

# Create a new study
fancy_new_study <- prolific_study(
    name = "A fancy study on Prolific",
    external_study_url = "https://www.my_fancy_study_url.com",
    completion_code = "123ab456cd78",
    eligibility_requirements = list(),
    estimated_completion_time = 1,
    reward = 1,
    total_available_places = 0
)

# Check the study's validity
```

```
print(fancy_new_study$validity_check())

# Whoops, better add a description and change the total_available_places,
# using RefClass and S4 methods for illustration
# both are equivalent, so only one of the two commands is required in practice
# RefClass variant
fancy_new_study$total_available_places <- 1L
# S4 variant
total_available_places(fancy_new_study) <- 1L

# RefClass variant
fancy_new_study$description <- "A fancy description"
# S4 variant
description(fancy_new_study) <- "A fancy description"

# Re-Check the study's validity
print(fancy_new_study$validity_check())

# Note: For the following code to work,
# you have to replace <api_token> in the code above by the actual API token

## Not run:
# Post the 'fancy_new_study' to Prolific - i.e. create it as a draft study on the platform
output_of_post <- prolific_api_access$access(
    endpoint = "studies",
    method = "post",
    data = fancy_new_study
)

# Success: fancy_new_study got an ID - it is now a draft study on Prolific!
fancy_new_study$id

# Note: The output of the access() command with a prolific_study object as `data` argument
# is a pointer to this prolific_study object.
# The prolific_study object is updated by reference
print(tracemem(output_of_post) == tracemem(fancy_new_study))

# Change the study's name
name(fancy_new_study) <- "A NEW name for 'fancy_new_study'"

# Update (patch) the study on Prolific,
# using S4 methods for illustration
output_of_patch <- access(
    prolific_api_access,
    endpoint = c("studies", id(fancy_new_study)),
    method = "patch",
    data = fancy_new_study
)

# Note: As above, the output of the access() command is a pointer to the prolific_study object.
print(tracemem(output_of_post) == tracemem(fancy_new_study))

# Delete fancy_new_study
```

```
prolific_api_access$access(
    endpoint = c("studies", id(fancy_new_study)),
    method = "delete",
    as_list = FALSE
)

## End(Not run)
```

Index