Package 'MTurkR'

August 29, 2016

Version 0.7.0

Date 2016-02-21

Title R Client for the MTurk Requester API

Maintainer Thomas J. Leeper <thosjleeper@gmail.com>

Imports utils, stats, curl, digest, base64enc, XML

Suggests testthat

Description Provides programmatic access to the Amazon Mechanical Turk (MTurk) Requester API.

License GPL-2

URL https://github.com/leeper/MTurkR

BugReports https://github.com/leeper/MTurkR/issues

NeedsCompilation no

Author Thomas J. Leeper [aut, cre],
Solomon Messing [ctb],
Sean Murphy [ctb],
Jonathan Chang [ctb]

Repository CRAN

Date/Publication 2016-02-22 01:32:05

R topics documented:

MTurkR-package *R Client for the MTurk Requester API*

Description

This package provides access to the Amazon Mechanical Turk (MTurk) Requester API. The package provides users of the MTurk Requester User Interface with access to a variety of functions currently unavailable to them (the creation and maintenance of worker Qualifications, email notifications to workers through `ContactWorker`, automated reviewing of assignments using Review Policies, and streamlined bonus payments through `GrantBonus`). It also provides users with all functions available in the RUI directly in R as well as a large number of other functions, and a simple, interactive command-line tool for performing many operations. A relatively fully featured, cross-platform graphical user interface is available in a separate add-on package called MTurkR-GUI.

Most users will find themselves using three principal functions: `CreateHIT`, `GetAssignments`, and `ApproveAssignments`, to create one or more HITs on the MTurk server, to retrieve completed assignments, and to approve assignments (and thus pay workers), respectively. As task complexity increases, additional functions are provided to handle worker qualifications, bonuses, emails to workers, automated review policies, bulk creation of HITs, and so forth.

Critically important, nothing in MTurkR will work during a given session without either first setting AWS credentials. The easiest way to do this is to specify '`AWS_ACCESS_KEY_ID`' and '`AWS_SECRET_ACCESS_KEY`' environment variables using `Sys.setenv()` or by placing these values in an .Renviron file. Credentials can also be specified atomically within each MTurkR function as the `keypair` argument, which accepts the credentials as a two-element character vector.

There are five common parameters that can be specified in most MTurkR functions: `keypair`, `verbose`, `log.requests`, and `sandbox`. The first of these is the AWS credentials parameter just described and the latter four are logicals. `verbose` causes certain information to be displayed on the standard output when functions are executed. `log.requests` records details of API calls in the working directory (see `readlogfile`). This is slightly time-consuming, so it can be omitted if you do not need access to log information (e.g., for troubleshooting). Setting the parameter `sandbox=TRUE` executes requests in the developer sandbox rather than the live server, which is a useful test environment.

All of these options can be set globally using `options()`. The specific forms are:

- `options('MTurkR.sandbox')`: A logical (default '`FALSE`'), to control use of the MTurk sandbox.
- `options('MTurkR.browser')`: A logical (default '`FALSE`'), to send requests to a web browser rather than through R.
- `options('MTurkR.verbose')`: A logical (default '`TRUE`'), to control printing to the console. If `TRUE`, updates will be printed to the console during function execution. This can be useful when, for example, executing a large number of assignment approvals, etc.
- `options('MTurkR.log')`: A logical (default '`TRUE`'), to control whether requests are logged to a directory.

- `options('MTurkR.logdir')`: A character string specifying a directory path in which to store the "MTurkRlog.tsv" log file. This is set, by default, as the working directory when MTurkR is loaded; if the option is not set (or unset), the current working directory is used.
- `options('MTurkR.test')`: A logical, which if `TRUE` (not the default), can allow `request` to be used for debugging purposes. It returns details of the request.

Author(s)

Thomas J. Leeper

Maintainer: Thomas J. Leeper <thosjleeper@gmail.com>

References

Amazon Mechanical Turk

The MTurkR Wiki, which contains numerous code examples and tutorials.

The MTurkRGUI Package, which provides a graphical user interface for MTurkR.

Amazon Mechanical Turk API Documentation

See Also

To get started using MTurkR, see the documentation for `CreateHIT` (for creating single tasks) and/or `BulkCreate` (for creating batches of tasks). For some tutorials on how to use MTurkR for specific use cases, see the following:

- survey, for collecting survey(-experimental) data
- categorization, for doing large-scale categorization (e.g., photo moderation or developing a training set)
- sentiment, for doing sentiment coding
- webscraping, for manual scraping of web data

`AccountBalance` *Retrieve MTurk account balance*

Description

Retrieves the amount of money (in US Dollars) in your MTurk account. `SufficientFunds` provides a wrapper that checks whether your account has sufficient funds based upon specified characters of your HIT.

Usage

```
AccountBalance(verbose = getOption('MTurkR.verbose', TRUE), ...)

SufficientFunds(amount = NULL, assignments = NULL, hits = NULL,
                bonus.ct = NULL, bonus.amount = NULL, masters = FALSE,
                turkfee = 0.2, turkmin = 0.01, mastersfee = 0.05, ...)
```

Arguments

amount	Intended per-assignment payment amount.
assignments	Number of intended assignments (per HIT, if multiple HITs).
hits	Number of HITs.
bonus.ct	Number of intended bonuses.
bonus.amount	Amount of each bonus.
masters	A logical indicating whether MTurk Masters will be used. Default is FALSE.
turkfee	Amazon's fee as percentage of payments. Default is 20-percent (as 0.20). Note, however, that MTurk charges an additional 20-percent if the number of assignments is greater than or equal to 10. This is factored in automatically
turkmin	Amazon's minimum per-assignment fee. Default is $0.01.
mastersfee	Amazon's additional charge for use of MTurk Masters. Default is 5-percent (as 0.05).
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

AccountBalance takes no substantive arguments. SufficientFunds is a wrapper for AccountBalance that accepts as inputs information about intended payments and bonuses to check whether your account has sufficient funds. If sandbox=TRUE, AccountBalance always returns "$10,000.00".

accountbalance() and getbalance() are aliases for AccountBalance.

Value

Return value is an object of class "MTurkResponse", including an additional character string (balance) containing the balance of the account in US Dollars. Note: object is returned invisibly.

Author(s)

Thomas J. Leeper

References

API Reference

MTurk Pricing Structure

Examples

```
## Not run:
AccountBalance()
SufficientFunds(amount = ".25", assignments = "50", hits = "5")
SufficientFunds(bonus.ct = "150", bonus.amount = ".75")

## End(Not run)
```

`ApproveAssignment` *Approve Assignment(s)*

Description

Approve one or more submitted assignments, or approve all assignments for a given HIT or HIT-Type. Also allows you to approve a previously rejected assignment. This function spends money from your MTurk account.

Usage

```
ApproveAssignment(assignments, feedback = NULL, rejected = FALSE,
                  verbose = getOption('MTurkR.verbose', TRUE), ...)

ApproveAllAssignments(hit = NULL, hit.type = NULL, annotation = NULL,
                      feedback = NULL,
                      verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`assignments`	A character string containing an AssignmentId, or a vector of multiple character strings containing multiple AssignmentIds, to approve.
`hit`	A character string containing a HITId all of whom's assignments are to be approved. Must specify `hit` xor `hit.type` xor `annotation`.
`hit.type`	A character string containing a HITTypeId (or a vector of HITTypeIds) all of whose HITs' assignments are to be approved. Must specify `hit` xor `hit.type` xor `annotation`.
`annotation`	An optional character string specifying the value of the `RequesterAnnotation` field for a batch of HITs. This can be used to approve all assignments for all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify `hit` xor `hit.type` xor `annotation`.
`feedback`	An optional character string containing any feedback for a worker. This must have length 1 or length equal to the number of workers. Maximum of 1024 characters. For `ApproveAllAssignments`, must be length 1.
`rejected`	A logical indicating whether the assignment(s) had previously been rejected (default `FALSE`). Approval of previously rejected assignments must be conducted separately from other approvals.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

Approve assignments, by AssignmentId (as returned by GetAssignment or by HITId or HITTypeId. Must specify assignments xor hit xor hit.type. ApproveAllAssignments approves all assignments of a given HIT or HITType without first having to perform GetAssignment.

ApproveAssignments() and approve() are aliases for ApproveAssignment. approveall() is an alias for ApproveAllAssignments.

Value

A dataframe containing the list of AssignmentIds, feedback (if any), and whether or not each approval request was valid.

Author(s)

Thomas J. Leeper

References

API Reference: Approve Assignment

API Reference: Approve Rejected Assignment

See Also

RejectAssignment

Examples

```
## Not run:
# Approve one assignment
ApproveAssignment(assignments = "26XXH0JPPSI23H54YVG7BKLEXAMPLE")

# Approve multiple assignments with the same feedback
ApproveAssignment(assignments = c("26XXH0JPPSI23H54YVG7BKLEXAMPLE1",
                                  "26XXH0JPPSI23H54YVG7BKLEXAMPLE2"),
                  feedback = "Great work!")


# Approve all assignments for a given HIT
ApproveAllAssignments(hit = "2MQB727M0IGF304GJ16S1F4VE3AYDQ")
# Approve all assignments for a given HITType
ApproveAllAssignments(hit.type = "2FFNCWYB49F9BBJWA4SJUNST5OFSOW")
# Approve all assignments for a given batch from the RUI
ApproveAllAssignments(annotation="BatchId:78382;")

## End(Not run)
```

AssignQualification *Assign Qualification*

Description

Assign a Qualification to one or more workers. The QualificationType should have already been created by CreateQualificationType, or the details of a new QualificationType can be specified atomically. This function also provides various options for automatically specifying the value of a worker's QualificationScore based upon a worker's statistics.

Usage

```
AssignQualification(qual, workers, value = "1", notify = FALSE,
                    name = NULL, description = NULL, keywords = NULL,
                    status = NULL, retry.delay = NULL,
                    test = NULL, answerkey = NULL, test.duration = NULL,
                    auto = NULL, auto.value = NULL,
                   conditional.statistic = NULL, conditional.comparator = NULL,
                    conditional.value = NULL, conditional.period = NULL,
                    set.statistic.as.value = FALSE,
                    verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual	A character string containing a QualificationTypeId.
workers	A character string containing a WorkerId, or a vector of character strings containing multiple WorkerIds.
value	A character string containing the value to be assigned to the worker(s) for the QualificationType.
notify	A logical indicating whether workers should be notified that they have been assigned the qualification. Default is FALSE.
name	An optional character string specifying a name for a new QualificationType. This is visible to workers. Cannot be modified by UpdateQualificationType.
description	An optional character string specifying a longer description of the QualificationType. This is visible to workers. Maximum of 2000 characters.
keywords	An optional character string containing a comma-separated set of keywords by which workers can search for the QualificationType. Cannot be modified by UpdateQualificationType. Maximum of 1000 characters.
status	A character vector of "Active" or "Inactive", indicating whether the QualificationType should be active and visible.
retry.delay	An optional time (in seconds) indicating how long workers have to wait before requesting the QualificationType after an initial rejection.
test	An optional character string consisting of a QuestionForm data structure, used as a test a worker must complete before the QualificationType is granted to them.

`answerkey` An optional character string consisting of an AnswerKey data structure, used to automatically score the test.

`test.duration` An optional time (in seconds) indicating how long workers have to complete the test.

`auto` A logical indicating whether the Qualification is automatically granted to workers who request it. Default is `FALSE`.

`auto.value` An optional parameter specifying the value that is automatically assigned to workers when they request it (if the Qualification is automatically granted).

`conditional.statistic`
An optional character string containing the name of a statistic (see `ListStatistics` that should be used to conditionally assign the QualificationType to workers.

`conditional.comparator`
An optional character string containing a comparator by which a worker's score of a qualification is compared to the specified `value`. One of "<", "<=", ">", ">=", "==", "!=", "Exists", or "DoesNotExist".

`conditional.value`
An optional numeric or character string value against which workers scores will be compared. The QualificationType will only be assigned to those whose score on the specified statistic meet the comparison to this value.

`conditional.period`
An optional character string specifying the period for the statistic. Must be one of: "OneDay", "SevenDays", "ThirtyDays", "LifeToDate". Default is "LifeToDate".

`set.statistic.as.value`
An optional logical specifying whether the worker's value of the statistic should be used as the value they are assigned for the QualificationType. Default is `FALSE` and `value` is used instead.

`verbose` Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.

`...` Additional arguments passed to `request`.

Details

A very robust function to assign a Qualification to one or more workers. The simplest use of the function is to assign a Qualification of the specified value to one worker, but assignment to multiple workers is possible. Workers can be assigned a Qualification previously created by `CreateQualificationType` or with the characteristics of a new QualificationType specified atomically. Qualifications can also be assigned conditional on each worker's value of a specified statistic (including assigning the value of the specified statistic as the worker's score for the Qualification).

`AssignQualifications()` and `assignqual()` are aliases.

Value

A dataframe containing the list of workers, the QualificationTypeId, the value each worker was assigned, whether they were notified of their QualificationType assignment, and whether the request was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

CreateQualificationType

UpdateQualificationScore

Examples

```
## Not run:
qual1 <-
CreateQualificationType(name="Worked for me before",
    description="This qualification is for people who have worked for me before",
    status = "Active",
    keywords="Worked for me before")

# assign qualification to single worker
AssignQualification(qual1$QualificationTypeId, "A1RO9UJNWXMU65", value = "50")

# assign a new qualification (defined atomically)
AssignQualification(workers = "A1RO9UJNWXMU65",
                    name = "Worked for me before",
                description = "This qualification is for people who have worked for me before",
                    status = "Active",
                    keywords = "Worked for me before")

# assign a qualification to a workers based upon their worker statistic
AssignQualification(qual1$QualificationTypeId,
    workers="A1RO9UJNWXMU65",
    conditional.statistic="NumberAssignmentsApproved",
    conditional.comparator=">",
    conditional.value="5",
    conditional.period="LifeToDate",
    set.statistic.as.value=TRUE)

DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

Blocking Workers *Block/Unblock Worker(s)*

Description

Block or unblock a worker. This prevents a worker from completing any HITs for you while they are blocked, but does not affect their ability to complete work for other requesters or affect their worker statistics. `GetBlockedWorkers` retrieves your list of currently blocked workers.

Usage

```
BlockWorker(workers, reasons, verbose = getOption('MTurkR.verbose', TRUE), ...)

UnblockWorker(workers, reasons = NULL, verbose = getOption('MTurkR.verbose', TRUE), ...)

GetBlockedWorkers(pagenumber = NULL, pagesize = NULL,
                  verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`workers`	A character string containing a WorkerId, or a vector of character strings containing multiple WorkerIds.
`reasons`	A character string containing a reason for blocking or unblocking a worker. This must have length 1 or length equal to the number of workers. It is required for `BlockWorker` and optional for `UnblockWorker`.
`pagenumber`	An optional integer (or character string) indicating which page of Blocked Workers search results should be returned. Most users can ignore this.
`pagesize`	An optional integer (or character string) indicating how many Blocked Workers should be returned per page of results. Most users can ignore this and the function will return the first 65,535 blocks.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

`BlockWorker` prevents the specified worker from completing any of your HITs. `UnblockWorker` reverses this operation.

`GetBlockedWorkers` retrieves currently blocked workers and the reason recorded for their block. This operation returns the first 65,535 blocked workers (the default for `pagesize`; access to additional blocked workers is available by specifying a `pagenumber` greater than 1.

`BlockWorkers()` and `block()` are aliases for `BlockWorker`. `UnblockWorkers()` and `unblock()` are aliases for `UnblockWorker`. `blockedworkers()` is an alias for `GetBlockedWorkers`.

Value

`BlockWorker` and `UnblockWorker` return a dataframe containing the list of workers, reasons (for blocking/unblocking them), and whether the request to block/unblock each of them was valid.

`GetBlockedWorkers` returns a dataframe containing the list of blocked workers and the recorded reason for the block.

Author(s)

Thomas J. Leeper

References

API Reference: Block

API Reference: Unblock

API Reference: GetBlockedWorkers

Examples

```
## Not run:

BlockWorker(worker, reasons="Did not follow photo categorization HIT instructions.")
GetBlockedWorkers()
UnblockWorker(worker)

## End(Not run)
```

BulkCreate *Generate Multiple HITs*

Description

Generate multiple HITs, possibly from an HTML template file, HITLayout parameters, or a vector of External Question URLs.

Usage

```
BulkCreate(questions, annotation, verbose = FALSE, ...)

BulkCreateFromTemplate(template, input, annotation, type = "HTMLQuestion",
                       frame.height = 450, verbose = FALSE, ...)

BulkCreateFromURLs(url, frame.height = 450, annotation, verbose = FALSE, ...)

BulkCreateFromHITLayout(hitlayoutid, input, annotation, verbose = FALSE, ...)
```

Arguments

questions	A character vector where each entry is a valid argument for the question argument to CreateHIT, or a list of objects of class "ExternalQuestion" (created by GenerateExternalQuestion) or HTMLQuestion (created by GenerateHTMLQuestion). Each entry in this vector or list represents one HIT to be created.
template	A character string or filename for a HIT template (probably a character string containing an HTML document or a path to a .html file).

input	A data.frame containing one row for each HIT to be created and columns named identically to the placeholders in the HIT template file (for BulkCreateFromTemplate) or the HITLayout parameters (for BulkCreateFromHITLayout). Operation will fail if variable names do not correspond.
type	A character string specifying how to wrap the resulting HIT question contents for use in CreateHIT. If set to "HTMLQuestion", template is passed to GenerateHTMLQuestion before use.
url	A character vector of URLs (served over HTTPS) of HIT files stored anywhere other than the MTurk server. See GenerateExternalQuestion.
frame.height	A character string containing the integer value (in pixels) of the frame height for the ExternalQuestion iframe. See GenerateExternalQuestion.
hitlayoutid	An optional character string including a HITLayoutId retrieved from a HIT "project" template generated in the Requester User Interface at 'https://requester.mturk.com/creat If the HIT template includes variable placeholders, must also specify hitlayoutparameters.
annotation	Either a one-element character vector containing a description for this group of HITs, or a character vector equal to the number of HITs to be created. This value is only visible to the requester. See CreateHIT for details.
verbose	Optionally print the results of the API request (and other details) to the standard output. Default is FALSE. Note that this overrides the default set by getOption('MTurkR.verbose') because in the case of many HITs, this output could become unwieldy.
...	Additional arguments passed to CreateHIT. See examples.

Details

These functions provide a wrapper for CreateHIT to be able to produce a group of HITs with identical properties. BulkCreateFromTemplate and BulkCreateFromHITLayout provide further wrappers that make it easy to create a group of HITs in a manner similar to using the Requester User Interface (RUI). BulkCreateFromURLs allows you to create multiple ExternalQuestion HITs.

The annotation field is required in order to group the HITs together and facilitate monitoring the group using other MTurkR functions. Note that these functions do not create a "batch" as used by the RUI; a batch can only be created through that interface.

Value

A list of data.frames, with each data.frame containing details of the HITs created. If all CreateHIT operations succeed, this response value can easily be collapsed into a single data.frame using do.call("rbind", value).

Author(s)

Thomas J. Leeper

References

API Reference: CreateHIT

Requester User Interface: HIT Template

API Reference: ExternalQuestion

See Also

CreateHIT

GenerateHITsFromTemplate

GenerateHITLayoutParameter

Examples

```
## Not run:
## BulkCreate ##

# load a vector of HTML files from the working directory
qvec <- sapply(list.files(pattern = ".html"), function(x) {
          paste0(readLines(x, warn = FALSE), collapse = "\n")
        })
# create a HIT from each question file
hits1 <- BulkCreate(questions = qvec,
                    annotation = paste("Bulk Create", Sys.Date()),
                    title = "Categorize an image",
                    description = "Categorize this image",
                    reward = ".05",
                    expiration = seconds(days = 4),
                    duration = seconds(minutes = 5),
                    keywords = "categorization, image, moderation, category")

# cleanup
ExpireHIT(annotation = paste("Bulk Create", Sys.Date()))
DisposeHIT(annotation = paste("Bulk Create", Sys.Date()))

## End(Not run)

## Not run:
## BulkCreateFromURLs ##

# create three HITs from the template
hits2 <-
BulkCreateFromURLs(url = paste0("https://www.example.com/",1:3,".html"),
                   frame.height = 400,
                   annotation = paste("Bulk From URLs", Sys.Date()),
                   title = "Categorize an image",
                   description = "Categorize this image",
                   reward = ".05",
                   expiration = seconds(days = 4),
                   duration = seconds(minutes = 5),
                   keywords = "categorization, image, moderation, category")

# cleanup
ExpireHIT(annotation = paste("Bulk From URLs", Sys.Date()))
DisposeHIT(annotation = paste("Bulk From URLs", Sys.Date()))

## End(Not run)
```

```
## Not run:
## BulkCreateFromTemplate ##

# load template HTML file
# should have placeholders of the form `${varName}` for variable values
temp <- system.file("templates/htmlquestion2.xml", package = "MTurkR")

# create/load data.frame of template variable values
a <- data.frame(hittitle = c("HIT title 1", "HIT title 2", "HIT title 3"),
                hitvariable = c("HIT text 1", "HIT text 2", "HIT text 3"),
                stringsAsFactors = FALSE)

# create three HITs from the template
hits3 <-
BulkCreateFromTemplate(template = temp,
                       input = a,
                       annotation = paste("Bulk From Template", Sys.Date()),
                       title = "Categorize an image",
                       description = "Categorize this image",
                       reward = ".05",
                       expiration = seconds(days = 4),
                       duration = seconds(minutes = 5),
                       keywords = "categorization, image, moderation, category")

# cleanup
ExpireHIT(annotation = paste("Bulk From Template", Sys.Date()))
DisposeHIT(annotation = paste("Bulk From Template", Sys.Date()))

## End(Not run)

## Not run:
## BulkCreateFromHITLayout ##

# retrieve HITLayoutID from Requester User Interface
layoutid <- "23ZGOOGQSCM61T1H5H9U0U00OQWFFU"

# create/load data.frame of HITLayout variable values
b <- data.frame(hittitle = c("HIT title 1", "HIT title 2", "HIT title 3"),
                hitvariable = c("HIT text 1", "HIT text 2", "HIT text 3"),
                stringsAsFactors = FALSE)

# create three HITs from the template
hits4 <-
BulkCreateFromHITLayout(hitlayoutid = layoutid,
                        input = b,
                        annotation = paste("Bulk From Layout", Sys.Date()),
                        title = "Categorize an image",
                        description = "Categorize this image",
                        reward = ".05",
                        expiration = seconds(days = 4),
                        duration = seconds(minutes = 5),
                        keywords = "categorization, image, moderation, category")
```

```
# cleanup
ExpireHIT(annotation = paste("Bulk From Layout", Sys.Date()))
DisposeHIT(annotation = paste("Bulk From Layout", Sys.Date()))

## End(Not run)
```

ChangeHITType *Change HITType Properties of a HIT*

Description

Change the HITType of a HIT from one HITType to another (e.g., to change the title, description, or qualification requirements associated with a HIT). This will cause a HIT to no longer be grouped with HITs of the previous HITType and instead be grouped with those of the new HITType. You cannot change the payment associated with a HIT without expiring the current HIT and creating a new one.

Usage

```
ChangeHITType(hit = NULL, old.hit.type = NULL, new.hit.type = NULL,
              title = NULL, description = NULL, reward = NULL, duration = NULL,
              keywords = NULL,
              auto.approval.delay = NULL, qual.req = NULL,
              old.annotation = NULL,
              verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`hit`	An optional character string containing the HITId whose HITTypeId is to be changed, or a vector of character strings containing each of multiple HITIds to be changed. Must specify `hit` xor `old.hit.type` xor `annotation`.
`old.hit.type`	An optional character string containing the HITTypeId (or a vector of HITTypeIds) whose HITs are to be changed to the new HITTypeId. Must specify `hit` xor `old.hit.type` xor `annotation`.
`new.hit.type`	An optional character string specifying the new HITTypeId that this HIT should be visibly grouped with (and whose properties, e.g. reward amount, this HIT should inherit).
`title`	An optional character string containing the title for the HITType. All HITs of this HITType will be visibly grouped to workers according to this title.
`description`	An optional character string containing a description of the HITType. This is visible to workers.
`reward`	An optional character string containing the per-assignment reward amount, in U.S. Dollars (e.g., "0.15").
`duration`	An optional character string containing the duration of each HIT, in seconds (for example, as returned by `seconds`).

`keywords` An optional character string containing a comma-separated set of keywords by which workers can search for HITs of this HITType.

`auto.approval.delay` An optional character string specifying the amount of time, in seconds (for example, as returned by `seconds`), before a submitted assignment is automatically granted.

`qual.req` An optional character string containing one a QualificationRequirement data structure, as returned by `GenerateQualificationRequirement`.

`old.annotation` An optional character string specifying the value of the `RequesterAnnotation` field for a batch of HITs to change the HITType of. This can be used to change the HITType for all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify `hit` xor `old.hit.type` xor `annotation`.

`verbose` Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.

`...` Additional arguments passed to `request`.

Details

This function changes the HITType of a specified HIT (or multiple specific HITs or all HITs of a specified HITType) to a new HITType. `hit` xor `old.hit.type` must be specified. Then, either a new HITTypeId can be specified or a new HITType can be created by atomically by specifying the characteristics of the new HITType.

`changehittype()` is an alias.

Value

A dataframe listing the HITId of each HIT who HITType was changed, its old HITTypeId and new HITTypeId, and whether the request for each HIT was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`CreateHIT`

`RegisterHITType`

Examples

```
## Not run:
hittype1 <-
RegisterHITType(title = "10 Question Survey",
                description =
                  "Complete a 10-question survey about news coverage and your opinions",
                reward = ".20",
                duration = seconds(hours=1),
                keywords = "survey, questionnaire, politics")
a <- GenerateExternalQuestion("http://www.example.com/", "400")
hit <- CreateHIT(hit.type = hittype1$HITTypeId,
                 assignments = 1,
                 expiration = seconds(days=1),
                 question = a$string)

# change to HITType with new reward amount
hittype2 <-
RegisterHITType(title = "10 Question Survey",
                description =
                  "Complete a 10-question survey about news coverage and your opinions",
                reward = ".45",
                duration = seconds(hours=1),
                keywords = "survey, questionnaire, politics")
ChangeHITType(hit = hit$HITId,
              new.hit.type=hittype2$HITTypeId)

# Change to new HITType, with arguments stated atomically
ChangeHITType(hit = hit$HITId,
              title = "10 Question Survey",
              description =
                "Complete a 10-question survey about news coverage and your opinions",
              reward = ".20",
              duration = seconds(hours=1),
              keywords = "survey, questionnaire, politics")

# expire and dispose HIT
ExpireHIT(hit = hit$HITId)
DisposeHIT(hit = hit$HITId)

## End(Not run)
```

`ContactWorker` *Contact Worker(s)*

Description

Contact one or more workers. This sends an email with specified subject line and body text to one or more workers. This can be used to recontact workers in panel/longitudinal research or to send follow-up work. Most likely will need to be used in tandem with `GrantBonus` to implement panels.

Usage

```
ContactWorker(subjects, msgs, workers, batch = FALSE,
              verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

subjects	A character string containing subject line of an email, or a vector of character strings of of length equal to the number of workers to be contacted containing the subject line of the email for each worker. Maximum of 200 characters.
msgs	A character string containing body text of an email, or a vector of character strings of of length equal to the number of workers to be contacted containing the body text of the email for each worker. Maximum of 4096 characters. Newlines can be specified with \n and tabs can be specified with \t in the message body.
workers	A character string containing a WorkerId, or a vector of character strings containing multiple WorkerIds.
batch	A logical (default is FALSE), indicating whether workers should be contacted in batches of 100 (the maximum allowed by the API). This significantly reduces the time required to contact workers, but eliminates the ability to send customized messages to each worker.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

Send an email to one or more workers, either with a common subject and body text or subject and body customized for each worker.

In batch mode (when batch=TRUE), workers are contacted in batches of 100 with a single identical email. If one email fails (e.g., for one worker) the other emails should be sent successfully. That is to say, the request as a whole will be valid but will return additional information about which workers were not contacted. This information can be found in the MTurkR log file and viewing the XML responses directly.

Note: It is only possible to contact workers who have performed work for you previously. When attempting to contact a worker who has not worked for you before, this function will indicate that the request was successful even though the email is not sent. The function will return a value of "HardFailure" for Valid when this occurs. The printed results may therefore appear contradictory because MTurk reports that requests to contact these workers are Valid, but they are not actually contacted. In batch, this means that a batch will be valid but individual ineligible workers will be reported as not contacted.

ContactWorkers() and contact() are aliases.

Value

A dataframe containing the list of workers, subjects, and messages, and whether the request to contact each of them was valid.

Author(s)

Thomas J. Leeper

References

API Reference

Examples

```
## Not run:
a <- "Complete a follow-up survey for $.50"
b <- "Thanks for completing my HIT!
I will pay a $.50 bonus if you complete a follow-up survey by Friday at 5:00pm.
The survey can be completed at
http://www.surveymonkey.com/s/pssurvey?c=A1RO9UEXAMPLE."

# contact one worker
c1 <- "A1RO9UEXAMPLE"
d <- ContactWorker(subjects = a,
                   msgs = b,
                   workers = c1)

# contact multiple workers in batch
c2 <- c("A1RO9EXAMPLE1","A1RO9EXAMPLE2","A1RO9EXAMPLE3")
e <- ContactWorker(subjects = a,
                   msgs = b,
                   workers = c2,
                   batch = TRUE)

## End(Not run)
```

`CreateHIT` *Create HIT*

Description

Create a single HIT. This is the most important function in the package. It creates a HIT based upon the specified parameters: (1) characteristics inherited from a HITType or specification of those parameters and (2) some kind of Question data structure. Use `BulkCreate` to create multiple HITs with shared properties.

Usage

```
CreateHIT(hit.type = NULL, question = NULL, validate.question = FALSE,
          expiration, assignments = "1",
          assignment.review.policy = NULL, hit.review.policy = NULL,
          annotation = NULL, unique.request.token = NULL,
          title = NULL, description = NULL, reward = NULL, duration = NULL,
          keywords = NULL,
```

```
            auto.approval.delay = NULL, qual.req = NULL,
            hitlayoutid = NULL, hitlayoutparameters = NULL,
          response.group = NULL, verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`hit.type`	An optional character string specifying the HITTypeId that this HIT should be visibly grouped with (and whose properties, e.g. reward amount, this HIT should inherit).
`question`	A mandatory (unless layoutid is specified) character string containing a QuestionForm, HTMLQuestion, or ExternalQuestion data structure. In lieu of a question parameter, a `hitlayoutid` and, optionally, `hitlayoutparameters` can be specified.
`validate.question`	A logical specifying whether the `question` parameter should be validated against the relevant MTurk schema prior to creating the HIT (operation will fail if it does not validate, and will return validation information). Default is `FALSE`.
`expiration`	The time (in seconds) that the HIT should be available to workers. Must be between 30 and 31536000 seconds.
`assignments`	A character string specifying the number of assignments
`assignment.review.policy`	An optional character string containing an Assignment-level ReviewPolicy data structure as returned by `GenerateAssignmentReviewPolicy`.
`hit.review.policy`	An optional character string containing a HIT-level ReviewPolicy data structure as returned by `GenerateHITReviewPolicy`.
`annotation`	An optional character string annotating the HIT. This is not visible to workers, but can be used as a label by which to identify the HIT from the API.
`unique.request.token`	An optional character string, included only for advanced users. It can be used to prevent creating a duplicate HIT. A HIT will not be creatd if a HIT was previously granted (within a short time window) using the same `unique.request.token`.
`title`	A character string containing the title for the HITType. All HITs of this HITType will be visibly grouped to workers according to this title. Maximum of 128 characters.
`description`	A character string containing a description of the HITType. This is visible to workers. Maximum of 2000 characters.
`reward`	A character string containing the per-assignment reward amount, in U.S. Dollars (e.g., "0.15").
`duration`	A character string containing the amount of time workers have to complete an assignment for HITs of this HITType, in seconds (for example, as returned by `seconds`). Minimum of 30 seconds and maximum of 365 days.
`keywords`	An optional character string containing a comma-separated set of keywords by which workers can search for HITs of this HITType. Maximum of 1000 characters.

`auto.approval.delay`
: An optional character string specifying the amount of time, in seconds (for example, as returned by `seconds`), before a submitted assignment is automatically granted. Maximum of 30 days.

`qual.req`
: An optional character string containing one or more QualificationRequirements data structures, for example as returned by `GenerateQualificationRequirement`.

`hitlayoutid`
: An optional character string including a HITLayoutId retrieved from a HIT "project" template generated in the Requester User Interface at 'https://requester.mturk.com/create
If the HIT template includes variable placeholders, must also specify `hitlayoutparameters`.

`hitlayoutparameters`
: An optional character string containing URL query parameter-formatted HITLayout parameters, for example returned by `GenerateHITLayoutParameter`. Must be specified along with a `hitlayoutid`.

`response.group`
: An optional character string (or vector of character strings) specifying what details of each HIT to return of: "Request", "Minimal", "HITDetail", "HITQuestion", "HITAssignmentSummary". For more information, see Common Parameters and HIT Data Structure.

`verbose`
: Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.

`...`
: Additional arguments passed to `request`.

Details

This function creates a new HIT and makes it available to workers. Characteristics of the HIT can either be specified by including a valid HITTypeId for "hit.type" or creating a new HITType by atomically specifying the characteristics of a new HITType.

When creating a HIT, some kind of Question data structure must be specified. Either, a QuestionForm, HTMLQuestion, or ExternalQuestion data structure can be specified for the `question` parameter or, if a HIT template created in the Requester User Interface (RUI) is being used, the appropriate `hitlayoutid` can be specified. If the HIT template contains variable placeholders, then the `hitlayoutparameters` should also be specified.

When creating a ExternalQuestion HITs, the `GenerateHITsFromTemplate` function can emulate the HIT template functionality by converting a template .html file into a set of individual HIT .html files (that would also have to be uploaded to a web server) and executing `CreateHIT` for each of these external files with an appropriate ExternalQuestion data structure specified for the `question` parameter.

`createhit()` and `create()` are aliases. `BulkCreate` can be used to create multiple HITs in a single call.

Value

A dataframe containing the HITId and other details of the newly created HIT.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

BulkCreate

ExtendHIT

ExpireHIT

DisableHIT

DisposeHIT

RegisterHITType

GenerateHITReviewPolicy

GenerateQualificationRequirement

Examples

```
## Not run:
## CreateHIT using HITLayout from MTurk Requester User Interface ##
a <- GenerateLayoutParameter("message","Text to display in HIT")
hit1 <- CreateHIT(hit.type = "2FFNCWYB49F9BBJWA4SJUNST5OFSOW",
                  hitlayoutid = "23ZGOOGQSCM61T1H5H9U0U00OQWFFU",
                  expiration = seconds(days = 4),
                  hitlayoutparameters = a)

## CreateHIT using ExternalQuestion HIT URL ##
eq <- GenerateExternalQuestion("https://www.example.com/","400")

### Specifying a HITTypeId ###
hit2 <- CreateHIT(hit.type = "2FFNCWYB49F9BBJWA4SJUNST5OFSOW",
                  expiration = seconds(days = 4),
                  question = eq$string)

### Creating a new HITTypeId atomically ###
hit3 <- CreateHIT(title = "Survey",
                  description = "5 question survey",
                  reward = ".10",
                  expiration = seconds(days = 4),
                  duration = seconds(hours = 1),
                  keywords = "survey, questionnaire",
                  question = eq$string)

## CreateHIT using HTMLQuestion HIT Contents ##
f1 <- system.file("templates/htmlquestion1.xml", package = "MTurkR")
hq <- GenerateHTMLQuestion(file = f1)
hit4 <- CreateHIT(hit.type = "2FFNCWYB49F9BBJWA4SJUNST5OFSOW",
                  expiration = seconds(days = 4),
                  question = hq$string)

## CreateHIT using QuestionForm HIT Contents ##
```

```
f2 <- system.file("templates/tictactoe.xml", package = "MTurkR")
qf <- GenerateHTMLQuestion(file = f2)
hit5 <- CreateHIT(hit.type = "2FFNCWYB49F9BBJWA4SJUNST5OFSOW",
                  expiration = seconds(days = 4),
                  question = qf$string)


## Cleanup examples ##
ExpireHIT(hit1$HITId)
ExpireHIT(hit2$HITId)
ExpireHIT(hit3$HITId)
ExpireHIT(hit4$HITId)
ExpireHIT(hit5$HITId)
DisposeHIT(hit1$HITId)
DisposeHIT(hit2$HITId)
DisposeHIT(hit3$HITId)
DisposeHIT(hit4$HITId)
DisposeHIT(hit5$HITId)

## End(Not run)
```

`CreateQualificationType` *Create QualificationType*

Description

Create a QualificationType. This creates a QualificationType, but does not assign it to any workers. All characteristics of the QualificationType (except name and keywords) can be changed later with `UpdateQualificationType`.

Usage

```
CreateQualificationType(name, description, status, keywords = NULL,
                        retry.delay = NULL, test = NULL, answerkey = NULL,
                        test.duration = NULL,
                        validate.test = FALSE, validate.answerkey = FALSE,
                        auto = NULL, auto.value = NULL,
                        verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`name`	A name for the QualificationType. This is visible to workers. It cannot be modified by `UpdateQualificationType`.
`description`	A longer description of the QualificationType. This is visible to workers. Maximum of 2000 characters.
`status`	A character vector of "Active" or "Inactive", indicating whether the QualificationType should be active and visible.

keywords	An optional character string containing a comma-separated set of keywords by which workers can search for the QualificationType. Maximum 1000 characters. These cannot be modified by UpdateQualificationType.
retry.delay	An optional time (in seconds) indicating how long workers have to wait before requesting the QualificationType after an initial rejection. If not specified, retries are disabled and Workers can request a Qualification of this type only once, even if the Worker has not been granted the Qualification.
test	An optional character string consisting of a QuestionForm data structure, used as a test a worker must complete before the QualificationType is granted to them.
answerkey	An optional character string consisting of an AnswerKey data structure, used to automatically score the test, perhaps as returned by GenerateAnswerKey.
test.duration	An optional time (in seconds) indicating how long workers have to complete the test.
validate.test	A logical specifying whether the test parameter should be validated against the relevant MTurk schema prior to creating the QualificationType (operation will fail if it does not validate, and will return validation information). Default is FALSE.
validate.answerkey	A logical specifying whether the answerkey parameter should be validated against the relevant MTurk schema prior to creating the QualificationType (operation will fail if it does not validate, and will return validation information). Default is FALSE.
auto	A logical indicating whether the Qualification is automatically granted to workers who request it. Default is NULL meaning FALSE.
auto.value	An optional parameter specifying the value that is automatically assigned to workers when they request it (if the Qualification is automatically granted).
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

A function to create a QualificationType. Active QualificationTypes are visible to workers and to other requesters. All characteristics of the QualificationType, other than the name and keywords, can later be modified by UpdateQualificationType. Qualifications can then be used to assign Qualifications to workers with AssignQualification and invoked as QualificationRequirements in RegisterHITType and/or CreateHIT operations.

createqual() is an alias.

Value

A dataframe containing the QualificationTypeId and other details of the newly created QualificationType.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GetQualificationType

DisposeQualificationType

UpdateQualificationType

SearchQualificationTypes

Examples

```
## Not run:
# Create
qual1 <- CreateQualificationType(name="Worked for me before",
           description="This qualification is for people who have worked for me before",
           status = "Active",
           keywords="Worked for me before")
DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)

## Not run:
# QualificationType with test and answer key
qf <- paste0(readLines(system.file("qualificationtest1.xml", package = "MTurkR")), collapse="")
qa <- paste0(readLines(system.file("answerkey1.xml", package = "MTurkR")), collapse="")
qual1 <- CreateQualificationType(name = "Qualification with Test",
           description = "This qualification is a demo",
           test = qf,
           answerkey = qa, # optionally, use AnswerKey
           status = "Active",
           keywords = "test, autogranted")
DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

`credentials` *Specify AWS Credentials*

Description

DEPRECATED

Usage

```
credentials(keypair = NULL)
```

Arguments

keypair	A two-item character vector containing the AWS Access Key ID and AWS Secret Access Key, in that order

Details

Do not use this function. It will be removed in the future.

The preferred way to load AWS credentials is now via environment variables. These environment variables can be set using `Sys.setenv()` or by using a local or global '`.Renviron`' file. To specify AWS credentials, set the '`AWS_ACCESS_KEY_ID`' and '`AWS_SECRET_ACCESS_KEY`' environment variables. These can be generated via the AWS IAM management console ('`https://console.aws.amazon.com/iam/home?#s`
Note that previously generated secret keys cannot be retrieved.

The `credentials()` function provides a legacy way of specify credentials and must be performed before any MTurk API requests can be successfully performed in a given session. The function simply stores the Access Key ID and the Secret Access Key as a two-item character vector in `getOption('MTurkR.keypair')`, which is called by default by all MTurkR operations. This operation can also be performed by loading `wizard.simple` or from within the MTurkRGUI package interface, which prompts for the keypair the first time it loads in a given R session.

Value

A two-item character vector containing an AWS Access Key ID in the first position and the corresponding Secret Access Key in the second position.

Author(s)

Thomas J. Leeper

DisableHIT *Disable HIT*

Description

Disabling a HIT is probably not what you want to do. DisableHIT automatically removes the HIT from the MTurk server, approves (and thus pays for) all submitted and pending assignments, and then permanently deletes all assignment data.

Usage

```
DisableHIT(hit = NULL, hit.type = NULL, annotation = NULL,
           response.group = NULL,
           verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`hit`	A character string containing a HITId or a vector of character strings containing multiple HITIds. Must specify `hit` xor `hit.type` xor `annotation`.
`hit.type`	An optional character string containing a HITTypeId (or a vector of HITTypeIds). Must specify `hit` xor `hit.type` xor `annotation`.
`annotation`	An optional character string specifying the value of the `RequesterAnnotation` field for a batch of HITs. This can be used to disable all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify `hit` xor `hit.type` xor `annotation`.
`response.group`	An optional character string specifying what details of each HIT to return of: "Minimal", "HITQuestion", "HITDetail", "HITAssignmentSummary". For more information, see Common Parameters and HIT Data Structure.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

Disable a HIT (and its assignment data). This is a somewhat risky function because it automatically approves all pending assignments and then disposes of everything. `DisposeHIT` is probably what most users will use to delete HIT and assignment data that is no longer needed.

`disable()` is an alias.

Value

A dataframe containing a list of HITs and whether the request to disable each of them was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`CreateHIT`

`ExtendHIT`

`ExpireHIT`

`DisposeHIT`

Examples

```
## Not run:
# Disable a single HIT
b <- GenerateExternalQuestion("http://www.example.com/","400")
hit1 <- CreateHIT(hit.type="2FFNCWYB49F9BBJWA4SJUNST5OFSOW",
                  expiration = seconds(days = 1),
                  question=b$string)
DisableHIT(hit = hit1$HITId)

# Disable all HITs of a given HITType
DisableHIT(hit.type = hit1$HITTypeId)

# Disable all HITs of a given batch from the RUI
DisableHIT(annotation="BatchId:78382;")

## End(Not run)
```

`DisposeHIT` *Dispose HIT*

Description

Dispose of a HIT that is no longer needed. You can only dispose of HITs that are Reviewable, with all assignments either approved or rejected.

Usage

```
DisposeHIT(hit = NULL, hit.type = NULL, annotation = NULL,
           response.group = NULL,
           verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`hit`	A character string containing a HITId or a vector of character strings containing multiple HITIds. Must specify `hit` xor `hit.type` xor `annotation`.
`hit.type`	An optional character string containing a HITTypeId (or a vector of HITTypeIds). Must specify `hit` xor `hit.type` xor `annotation`.
`annotation`	An optional character string specifying the value of the `RequesterAnnotation` field for a batch of HITs. This can be used to dispose of all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify `hit` xor `hit.type` xor `annotation`.
`response.group`	An optional character string specifying what details of each HIT to return of: "Minimal", "HITQuestion", "HITDetail", "HITAssignmentSummary". For more information, see Common Parameters and HIT Data Structure.

verbose	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
...	Additional arguments passed to `request`.

Details

Dispose of a HIT (and its assignment data) when it is no longer needed. Must specify a HITId or a HITTypeId, but not both. HITTypeId uses the `SearchHITs` operation to locate HITs of the specified HITType before disposing of them.

`disposehit()` is an alias.

Value

A dataframe containing a list of HITs and whether the request to dispose of each of them was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`CreateHIT`

`ExtendHIT`

`ExpireHIT`

`DisableHIT`

Examples

```
## Not run:
# Dispose a single HIT
b <- GenerateExternalQuestion("http://www.example.com/","400")
hit1 <-
CreateHIT(hit.type = "2FFNCWYB49F9BBJWA4SJUNST5OFSOW",
          expiration = seconds(days = 1),
          question=b$string)
ExpireHIT(hit1$HITId) # must be expired before disposing
DiposeHIT(hit1$HITId)

# Dispose all HITs of a given HITType
DisposeHIT(hit.type = hit1$HITTypeId)

# Dispose all HITs of a given batch from the RUI
DisposeHIT(annotation="BatchId:78382;")

## End(Not run)
```

DisposeQualificationType *Dispose QualificationType*

Description

Dispose of a QualificationType. This deletes the QualificationType, Qualification scores for all workers, and all records thereof.

Usage

```
DisposeQualificationType(qual, verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual	A character string containing a QualificationTypeId.
verbose	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
...	Additional arguments passed to `request`.

Details

A function to dispose of a QualificationType that is no longer needed. All information about the QualificationType and all workers' Qualifications of that type are permanently deleted.

`disposequal()` is an alias.

Value

A dataframe containing the QualificationTypeId and whether the request to dispose was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`GetQualificationType`

`CreateQualificationType`

`UpdateQualificationType`

`SearchQualificationTypes`

Examples

```
## Not run:
qual1 <-
CreateQualificationType(name="Worked for me before",
    description="This qualification is for people who have worked for me before",
    status = "Active",
    keywords="Worked for me before")
DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

`ExpireHIT` *Expire HIT*

Description

Force a HIT to expire immediately, as opposed to at its prespecified expiration time. Expired HITs can be extended with the `ExtendHIT` operation.

Usage

```
ExpireHIT(hit = NULL, hit.type = NULL, annotation = NULL,
          verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`hit`	A character string containing a HITId or a vector of character strings containing multiple HITIds. Must specify `hit` xor `hit.type` xor `annotation`, otherwise all HITs are returned in `HITStatus`.
`hit.type`	An optional character string containing a HITTypeId (or a vector of HITTypeIds). Must specify `hit` xor `hit.type` xor `annotation`, otherwise all HITs are returned in `HITStatus`.
`annotation`	An optional character string specifying the value of the `RequesterAnnotation` field for a batch of HITs. This can be used to expire all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify `hit` xor `hit.type` xor `annotation`, otherwise all HITs are returned in `HITStatus`.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

A function to (prematurely) expire a HIT (or multiple HITs), thereby preventing any additional assignments from being completed. Pending assignments can still be submitted. An expired HIT can be reactived by adding additional time to its expiration using `ExtendHIT`.

`expire()` is an alias.

Value

A dataframe containing the HITId(s) and whether each expiration request was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

CreateHIT

ExtendHIT

DisableHIT

DisposeHIT

Examples

```
## Not run:
a <- GenerateExternalQuestion("http://www.example.com/","400")
hit1 <-
CreateHIT(hit.type="2FFNCWYB49F9BBJWA4SJUNST5OFSOW", question = a$string)

# expire HIT
ExpireHIT(hit = hit1$HITId)

# Expire all HITs of a given batch from the RUI
ExpireHIT(annotation="BatchId:78382;")

## End(Not run)
```

ExtendHIT *Extend HIT*

Description

Extend the time remaining on a HIT or the number of assignments available for the HIT.

Usage

```
ExtendHIT(hit = NULL, hit.type = NULL, annotation = NULL,
          add.assignments = NULL, add.seconds = NULL,
        unique.request.token = NULL, verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`hit`	An optional character string containing a HITId or a vector of character strings containing multiple HITIds. Must specify `hit` xor `hit.type` xor `annotation`.
`hit.type`	An optional character string containing a HITTypeId (or a vector of HITTypeIds). Must specify `hit` xor `hit.type` xor `annotation`.
`annotation`	An optional character string specifying the value of the `RequesterAnnotation` field for a batch of HITs. This can be used to extend all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify `hit` xor `hit.type` xor `annotation`.
`add.assignments`	An optional character string containing the number of assignmetns to add to the HIT. Must be between 1 and 1000000000.
`add.seconds`	An optional character string containing the amount of time to extend the HIT, in seconds (for example, returned by `seconds`). Must be between 1 hour (3600 seconds) and 365 days.
`unique.request.token`	An optional character string, included only for advanced users.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

A useful function for adding time and/or additional assignments to a HIT. If the HIT is already expired, this reactivates the HIT for the specified amount of time. If all assignments have already been submitted, this reactivates the HIT with the specified number of assignments and previously specified expiration. Must specify a HITId xor a HITTypeId. If multiple HITs or a HITTypeId are specified, each HIT is extended by the specified amount.

`extend()` is an alias.

Value

A dataframe containing the HITId, assignment increment, time increment, and whether each extension request was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

CreateHIT

ExpireHIT

DisableHIT

DisposeHIT

Examples

```
## Not run:
a <- GenerateExternalQuestion("https://www.example.com/","400")
hit1 <- CreateHIT(title = "Example",
                  description = "Simple Example HIT",
                  reward = ".01",
                  expiration = seconds(days = 4),
                  duration = seconds(hours = 1),
                  keywords = "example",
                  question = a$string)

# add assignments
ExtendHIT(hit = hit1$HITId, add.assignments = "20")

# add time
ExtendHIT(hit = hit1$HITId, add.seconds = seconds(days=1)))

# add assignments and time
ExtendHIT(hit = hit1$HITId, add.assignments = "20", add.seconds = seconds(days=1)))

# cleanup
DisableHIT(hit = hit1$HITId)


## End(Not run)
## Not run:
# Extend all HITs of a given batch from the RUI
ExtendHIT(annotation="BatchId:78382;", add.assignments = "20")

## End(Not run)
```

GenerateAnswerKey *Generate AnswerKey Data Structure*

Description

Generate an AnswerKey data structure for a Qualification test.

Usage

```
GenerateAnswerKey(questions, scoring = NULL)
AnswerKeyTemplate(xml.parsed = NULL)
```

Arguments

questions	A dataframe containing QuestionIdentifiers, AnswerOptions, AnswerScores, and DefaultScores. See MTurk API Documentation.
scoring	An optional list containing QualificationValueMapping information. See MTurk API Documentation.
xml.parsed	A complete QuestionForm data structure parsed by xmlParse. Must specify this or the xml parameter.

Details

GenerateAnswerKey creates an AnswerKey data structure (possibly from a template created by AnswerKeyTemplate from a QuestionForm data structure), which serves to automatically score a Qualification test, as specified in the test parameter of CreateQualificationType. An AnswerKey data structure is also returned by GetQualificationType.

Value

GenerateAnswerKey returns a list containing an AnswerKey data structure as a parsed XML tree, character string containing that tree, and a url-encoded character string.

AnswerKeyTemplate returns a list that can be used in the questions parameter of GenerateAnswerKey. Placeholders are left for AnswerScore values to be manually entered prior to using it in GenerateAnswerKey.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

CreateQualificationType

Examples

```
## Not run:
# generate an AnswerKey from a list of arguments
qs <- list(list(QuestionIdentifier = "Question1",
                AnswerOption = list(SelectionIdentifier="A", AnswerScore=15),
                AnswerOption = list(SelectionIdentifier="B", AnswerScore=10),
                DefaultScore = 5),
           list(QuestionIdentifier = "Question2",
                AnswerOption = list(SelectionIdentifier="D", AnswerScore=10) ) )

scoring1 <- list(PercentageMapping=5)

scoring2 <- list(RangeMapping=list(list(InclusiveLowerBound=0,
```

```
                                        InclusiveUpperBound=20,
                                        QualificationValue=5),
                                    list(InclusiveLowerBound=21,
                                        InclusiveUpperBound=100,
                                        QualificationValue=10)),
                    OutOfRangeQualificationValue=0)

ak1 <- GenerateAnswerKey(qs, scoring1)
ak2 <- GenerateAnswerKey(qs, scoring2)

# generate an AnswerKey template from a QualificationTest
qt <- system.file("templates", "qualificationtest1.xml", package = "MTurkR")
akt <- AnswerKeyTemplate(xmlParse(qt))
# use the template to generate an AnswerKey
ak <- GenerateAnswerKey(ak)

## End(Not run)
```

`GenerateExternalQuestion`
Generate ExternalQuestion

Description

Generate an ExternalQuestion data structure for use in the 'Question' parameter of the `CreateHIT` operation.

Usage

```
GenerateExternalQuestion(url, frame.height = 400)
```

Arguments

`url`	A character string containing the URL (served over HTTPS) of a HIT file stored anywhere other than the MTurk server.
`frame.height`	A character string containing the integer value (in pixels) of the frame height for the ExternalQuestion iframe.

Details

An ExternalQuestion is a HIT stored anywhere other than the MTurk server that is displayed to workers within an HTML iframe of the specified height. The URL should point to a page — likely an HTML form — that can retrieve several URL GET parameters for "AssignmentId" and "WorkerId", which are attached by MTurk when opening the URL. The page should also be able to submit those parameters plus any assignment data to `https://www.mturk.com/mturk/externalSubmit` (for the live MTurk site) or `https://workersandbox.mturk.com/mturk/externalSubmit` (for the sandbox site), using either the HTTP GET or POST methods.

Note: `url` must be HTTPS. See Wikipedia:HTTP Secure for details.

Value

A list containing `xml.parsed`, an XML data structure, `string`, xml formatted as a character string, and `url.encoded`, character string containing a URL query parameter-formatted HTMLQuestion data structure for use in the `question` parameter of `CreateHIT`.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`CreateHIT`

`GenerateHITLayoutParameter`

Examples

```
## Not run:
a <- GenerateExternalQuestion(url="http://www.example.com/", frame.height="400")

hit1 <-
CreateHIT(title = "Survey",
          description = "5 question survey",
          reward = ".10",
          expiration = seconds(days = 4),
          duration = seconds(hours = 1),
          keywords = "survey, questionnaire",
          question = a$string)

ExpireHIT(hit1$HITId)
DisposeHIT(hit1$HITId)

## End(Not run)
```

`GenerateHITLayoutParameter`

Generate a HITLayout Parameter

Description

Generate a HITLayout parameter based upon the names of HIT template variables and the values to substitute for those variables in a single HIT. Used in collaboration with a HIT Layout ID from '`https://requester.mturk.com/create/projects`' in the `CreateHIT` operation.

Usage

```
GenerateHITLayoutParameter(names, values)
```

Arguments

names	A character string containing the name of a HIT template variable or a vector of character strings containing the names of multiple HIT template variables. This is optional if values has a non-empty names attribute.
values	A character string containing the value of a HIT template variable to be inserted for a specific HIT or a vector of character strings containing the values of multiple HIT template variables to be inserted for a specific HIT. If values has a non-empty names attribute and names is missing, the names attribute is used.

Details

This function provides the content for the hitlayoutparameters option of CreateHIT. Specifically, a HIT Template created in the MTurk Requester User Interface (RUI) has a number of placeholder variables for content to be inserted. This fucntion supplies the content to be inserted into the template for one HIT. If multiple HITs are being created from one template, then GenerateHITLayoutParameter should be run once for each HIT.

Analogous functionality for producing .html files on the local workstation (e.g., to create multiple external HITs from the same template) is provided by GenerateHITsFromTemplate.

Value

A character string containing URL query parameter-formatted HITLayout parameters, to be used in the hitlayoutparameters parameter of CreateHIT.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

CreateHIT

GenerateExternalQuestion

GenerateHITsFromTemplate

Examples

```
## Not run:
# examples of specifying 'names' and 'values'
a <- GenerateHITLayoutParameter(names = "hitvariable",
                                values = "Text for HIT 1")
b <- GenerateHITLayoutParameter(names = "hitvariable",
```

```
                                values = "Text for HIT 2")
c <- GenerateHITLayoutParameter(names = c("hitvariable1","hitvariable2"),
                                values = c("Headline for HIT1","Text for HIT 1"))

# example using a named character string in lieu of specifying 'names'
d <- GenerateHITLayoutParameter(values = c(hitvariable1 = "Headline for HIT1",
                                           hitvariable2 = "Text for HIT 1"))

# create HIT using layout parameter
hit1 <-
CreateHIT(title = "Survey",
          description = "5 question survey",
          reward = ".10",
          expiration = seconds(days=4),
          duration = seconds(hours = 1),
          keywords = "survey, questionnaire",
            # retrieved from MTurk web interface:
          hitlayoutid = "23ZGOOGQSCM61T1H5H9U0U00OQWFFU",
          hitlayoutparameters = a)

# cleanup
DisableHIT(hit1$HITId)

## End(Not run)
```

`GenerateHITsFromTemplate`

Generate HITs from a Template

Description

Generate individual HIT .html files from a local .html HIT template file, in the same fashion as the MTurk Requester User Interface (RUI).

Usage

```
GenerateHITsFromTemplate(template, input, filenames = NULL, write.files = FALSE)
```

Arguments

`template`	A character string or filename for an .html HIT template
`input`	A data.frame containing one row for each HIT to be created and columns named identically to the placeholders in the HIT template file. Operation will fail if variable names do not correspond.
`filenames`	An optional list of filenames for the HITs to be created. Must be equal to the number of rows in `input`.
`write.files`	A logical specifying whether HIT .html files should be created and stored in the working directory. Or, alternatively, whether HITs should be returned as character vectors in a list.

... Additional arguments passed to CreateHIT.

Details

GenerateHITsFromTemplate generates individual HIT question content from a HIT template (containing placeholders for input data of the form ${variablename}). The tool provides functionality analogous to the MTurk RUI HIT template and can be performed on .html files generated therein. The HITs are returned as a list of character strings. If write.files = TRUE, a side effect occurs in the form of one or more .html files being written to the working directory, with filenames specified by the filenames option or, if filenames=NULL of the form "NewHIT1.html", "NewHIT2.html", etc.

Value

A list containing a character string for each HIT generated from the template.

Author(s)

Thomas J. Leeper

References

API Reference: Operation

API Reference: ExternalQuestion Data Strucure

See Also

BulkCreateFromTemplate

Examples

```
## Not run:
# create/edit template HTML file
# should have placeholders of the form `${varName}` for variable values
temp <- system.file("templates/htmlquestion2.xml", package = "MTurkR")
readLines(temp)

# create/load data.frame of template variable values
a <- data.frame(hittitle = c("HIT title 1","HIT title 2","HIT title 3"),
                hitvariable = c("HIT text 1","HIT text 2","HIT text 3"),
                stringsAsFactors=FALSE)

# create HITs from template and data.frame values
temps <- GenerateHITsFromTemplate(template = temp, input = a)

# create HITs from template
hittype1 <- RegisterHITType(title = "2 Question Survey",
              description = "Complete a 2-question survey",
              reward = ".20",
              duration = seconds(hours=1),
              keywords = "survey, questionnaire, politics")
```

```
hits <- lapply(temps, function(x) {
    CreateHIT(hit.type = hittype1$HITTypeId,
              expiration = seconds(days = 1),
              assignments = 2,
              question = GenerateHTMLQuestion(x)$string)
})

# cleanup
ExpireHIT(hit.type = hittype1$HITTypeId)
DisposeHIT(hit.type = hittype1$HITTypeId)

## End(Not run)
```

`GenerateHTMLQuestion` *Generate HTMLQuestion*

Description

Generate an HTMLQuestion data structure for use in the 'Question' parameter of `CreateHIT`.

Usage

```
GenerateHTMLQuestion(character = NULL, file = NULL, frame.height = 450)
```

Arguments

`character`	An optional character string from which to construct the HTMLQuestion data structure.
`file`	An optional character string containing a filename from which to construct the HTMLQuestion data structure.
`frame.height`	A character string containing the integer value (in pixels) of the frame height for the HTMLQuestion iframe.

Details

Must specify either `character` or `file`.

To be valid, an HTMLQuestion data structure must be a complete XHTML document, including doctype declaration, head and body tags, and a complete HTML form (including the `form` tag with a submit URL, the assignmentId for the assignment as a form field, at least one substantive form field (can be hidden), and a submit button that posts to the external submit URL; see `GenerateExternalQuestion`.). If you fail to include a complete `form`, workers will be unable to submit the HIT. See the API Documentation for a complete example.

MTurkR comes pre-installed with several simple examples of HTMLQuestion HIT templates, which can be found by examing the '`templates`' directory of the installed package directory. These examples include simple HTMLQuestion forms, as well as templates for categorization, linking to off-site surveys, and sentiment analysis. Note that the examples, while validated complete, do not include CSS styling.

Value

A list containing `xml.parsed`, an XML data structure, `string`, xml formatted as a character string, and `url.encoded`, character string containing a URL query parameter-formatted HTMLQuestion data structure for use in the `question` parameter of `CreateHIT`.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`CreateHIT`

`GenerateExternalQuestion`

`GenerateHITLayoutParameter`

Examples

```
## Not run:
f <- system.file("templates/htmlquestion1.xml", package = "MTurkR")
a <- GenerateHTMLQuestion(file=f)

hit1 <-
CreateHIT(title = "Survey",
          description = "5 question survey",
          reward = ".10",
          expiration = seconds(days = 4),
          duration = seconds(hours = 1),
          keywords = "survey, questionnaire",
          question = a$string)

ExpireHIT(hit1$HITId)
DisposeHIT(hit1$HITId)

## End(Not run)
```

`GenerateNotification` *Generate Notification*

Description

Generate a HITType Notification data structure for use in `SetHITTypeNotification`.

Usage

```
GenerateNotification(destination, transport = "Email", event.type,
                     version = "2006-05-05", event.number = "1")
```

Arguments

destination	Currently, a character string containing a complete email address (if transport="Email") or the SQS URL (if transport="SQS").
transport	Currently only "Email" and "SQS" are supported. AWS recommends the use of the SQS transport.
event.type	A character string containing one of: AssignmentAccepted, AssignmentAbandoned, AssignmentReturned, AssignmentSubmitted, HITReviewable, HITExpired, or Ping.
version	Version of the HITType Notification API to use. Intended only for advanced users.
event.number	Intended only for advanced users to construct custom Notifications.

Details

Generate a Notification data structure for use in the notification option of SetHITTypeNotification.

Value

A character string containing a URL query parameter-formatted Notification data structure.

Author(s)

Thomas J. Leeper

References

API Reference

API Reference: Concept

See Also

SetHITTypeNotification

SendTestEventNotification

GenerateQualificationRequirement
Generate QualificationRequirement

Description

Generate a QualificationRequirement data structure for use with CreateHIT or RegisterHITType.

Usage

```
GenerateQualificationRequirement(qual,
                                 comparator,
                                 value,
                                 preview = NULL,
                                 qual.number = NULL)
```

Arguments

qual	A character string containing a QualificationTypeId, or a vector of QualificationTypeIds. This parameter also accepts shorthand labels for built-in QualificationTypes: "Approved" (percent of assignments approved), "NumberApproved" (number of assignments approved), "Locale", "Adult", and MTurk "masters" QualificationTypes ("Masters").
comparator	A character string containing a comparator, or a vector of comparators, by which a worker's score of a qualification is compared to the specified value. One of <,<=,>,>=,==,!=,"Exists","DoesNotExist","In","NotIn". For "Masters"-type qualifications, only "Exists" and "DoesNotExist" are available.
value	A numeric or character string value (or vector of such) against which workers scores will be compared. Must be a non-negative integer, except when qualification="Locale" (when it must be a two-digit country code, or a five-character ISO 3166-2 identifier for a U.S. state of the form US-MN using the two-letter abbreviation for Minnesota, etc.) or when comparator is "Exists" or "DoesNotExist" (when it must be an empty character string). When using the "In" or "NotIn" comparators, each element in value can be a comma-separated string of up to 15 values (e.g., 15 discrete scores or 15 discrete locales to use for that Qualification).
preview	An optional logical specifying whether a worker must have the Qualification in order to preview the HIT on the MTurk worker site. The default is FALSE.
qual.number	An argument for primarily internal use.

Details

A convenience function to translate the details of a QualificationRequirement into the necessary structure for use in the qual.req parameter of CreateHIT or RegisterHITType. The function accepts three required parameters: qual, comparator, and value. qual must be a valid QualificationTypeId for either a built-in QualificaitonType (see ListQualificationTypes) or a custom

QualificationType (e.g., one created with CreateQualificationType). Multiple QualificationRequirements can be generated in one call — that is, if a requester intends to impose multiple QualificationRequirements on a single HITType, those requirements must be specified in a single call to GenerateQualificationRequirements. Once attached to a HITType, only workers who meet all of the specified QualificationRequirements can complete assignments for a HIT of that HITType.

Value

Returns a character string (of class "QualificationRequirement") containing URL-encoded QualificationRequirements.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

CreateHIT

RegisterHITType

Examples

```
a <- ListQualificationTypes()[2,2] # Number of HITs Approved
# one QualificationRequirement
q1 <- GenerateQualificationRequirement(a,">","90")
# two QualificationRequirements
q2 <- GenerateQualificationRequirement(c("Locale","Approved"),
                                       c("==",">"),
                                       c("US",90),
                                       preview = TRUE)
# one QualificationRequirement using the 'In' comparator
q3 <- GenerateQualificationRequirement("Locale","In","US,GB")
# two QualificationRequirements using the 'In' comparator
q4 <- GenerateQualificationRequirement(c("Locale","Approved"),
                                       c("==",">"),
                                       c("US,GB,DK",90),
                                       preview = c(TRUE,FALSE))
# `Exists` comparator
q5 <- GenerateQualificationRequirement("Approved", "Exists", "")

# Masters `DoesNotExist` comparator
q6 <- GenerateQualificationRequirement("Masters", "DoesNotExist", "")

# U.S. state locale value
q7a <- GenerateQualificationRequirement("Locale","==","US-MN")
## with multiple states and an 'In' operator
q7b <- GenerateQualificationRequirement("Locale","In","US-MN,US-IL")
```

```
# Complex locale values
q8 <- GenerateQualificationRequirement("Locale","In","US-NY,US-WA,CA")
```

`GenerateReviewPolicy` *Generate HIT and/or Assignment ReviewPolicies*

Description

Generate a HIT ReviewPolicy and/or Assignment ReviewPolicy data structure for use in `CreateHIT`.

Usage

```
GenerateHITReviewPolicy(...)

GenerateAssignmentReviewPolicy(...)
```

Arguments

`...`	ReviewPolicy parameters passed as named arguments. See details and examples.

Details

Converts a list of ReviewPolicy parameters into a ReviewPolicy data structure.

A ReviewPolicy works by testing whether an assignment or a set of assignments satisfies a particular condition. If that condition is satisfied, then specified actions are taken. ReviewPolicies come in two "flavors": Assignment-level ReviewPolicies take actions based on "known" answers to questions in the HIT and HIT-level ReviewPolicies take actions based on agreement among multiple assignments. It is possible to specify both Assignment-level and HIT-level ReviewPolicies for the same HIT.

Assignment-level ReviewPolicies involve checking whether that assignment includes particular ("correct") answers. For example, an assignment might be tested to see whether a correct answer is given to one question by each worker as a quality control measure. The ReviewPolicy works by checking whether a specified percentage of known answers are correct. So, if a ReviewPolicy specifies two known answers for a HIT and the worker gets one of those known answers correct, the ReviewPolicy scores the assignment at 50 (i.e., 50 percent). The ReviewPolicy can then be customized to take three kinds of actions depending on that score: `ApproveIfKnownAnswerScoreIsAtLeast` (approve the assignment automatically), `RejectIfKnownAnswerScoreIsLessThan` (reject the assignment automatically), and `ExtendIfKnownAnswerScoreIsLessThan` (add additional assignments and/or time to the HIT automatically). The various actions can be combined to, e.g., both reject an assignment and add further assignments if a score is below the threshold, or reject below a threshold and approve above, etc.

HIT-level ReviewPolicies involve checking whether multiple assignments submitted for the same HIT "agree" with one another. Agreement here is very strict: answers must be exactly the same

across assignments for them to be a matched. As such, it is probably only appropriate to use closed-ended (e.g., multiple choice) questions for HIT-level ReviewPolicies otherwise ReviewPolicy actions might be taken on irrelevant differences (e.g., word capitalization, spacing, etc.). The ReviewPolicy works by checking whether answers to multiple assignments are the same (or at least whether a specified percentage of answers to a given question are the same). For example, if the goal is to categorize an image into one of three categories, the ReviewPolicy will check whether two of three workers agree on the categorization (known as the "HIT Agreement Score", which is a percentage of all workers who agree). Depending on the value of the HIT Agreement Score, actions can be taken. As of October 2014, only one action can be taken: `ExtendIfHITAgreementScoreIsLessThan` (extending the HIT in assignments by the number of assignments specified in `ExtendMaximumAssignments` or time as specified in `ExtendMinimumTimeInSeconds`).

Another agreement score (the "Worker Agreement Score"), measured the percentage of a worker's responses that agree with other workers' answers. Depending on the Worker Agreement Score, two actions can be taken: `ApproveIfWorkerAgreementScoreIsAtLeast` (to approve the assignment automatically) or `RejectIfWorkerAgreementScoreIsLessThan` (to reject the assignment automatically, with an optional reject reason supplied with `RejectReason`). A logical value (`DisregardAssignmentIfRejected`) specifies whether to exclude rejected assignments from the calculation of the HIT Agreement Score.

Note: An optional `DisregardAssignmentIfKnownAnswerScoreIsLessThan` excludes assignments if those assignments score below a specified "known" answers threshold as determined by a separate Assignment-level ReviewPolicy.

Value

A character string that reflects the URL-encoded `HITReviewPolicy` or `AssignmentReviewPolicy`.

Author(s)

Thomas J. Leeper

References

API Reference: QuestionForm

API Reference (ReviewPolicies)

API Reference (Data Structure)

See Also

`CreateHIT`

`GetReviewResultsForHIT`

Examples

```
## Not run:
# HIT-level ReviewPolicies #

## Conditionally extend HIT based on HIT Agreement Score
lista <- list(QuestionIds = c("Question1","Question2","Question5"),
              QuestionAgreementThreshold = 49, # at least 50 percent agreement
              ExtendIfHITAgreementScoreIsLessThan = 50,
```

```
              ExtendMinimumTimeInSeconds = 3600,
              ExtendMaximumAssignments = 2,
              DisregardAssignmentIfRejected = TRUE)
policya <- do.call(GenerateHITReviewPolicy, lista)

## Conditionally approve and reject based on Worker Agreement Score
listb <- list(QuestionIds = c("Question1","Question2","Question5"),
              QuestionAgreementThreshold = 65, # at least two of three 'correct' answers
              ApproveIfWorkerAgreementScoreIsAtLeast = 65,
              RejectIfWorkerAgreementScoreIsLessThan = 34,
              DisregardAssignmentIfRejected = TRUE)
policyb <- do.call(GenerateHITReviewPolicy, listb)

# Attach an assignment review policy to a HIT
hit1 <-
CreateHIT(title = "Survey",
          description = "5 question survey",
          reward = ".10",
          expiration = seconds(days = 4),
          duration = seconds(hours = 1),
          keywords = "survey, questionnaire",
          hit.review.policy = policyb,
          question = GenerateExternalQuestion("http://www.example.com/","400"))

# GetReviewResults
GetReviewResultsForHIT(hit1$HITId)

# cleanup
ExpireHIT(hit1$HITId)
DisposeHIT(hit1$HITId)

## End(Not run)
## Not run:
# Assignment-level ReviewPolicies #

# Example Policy A
## Conditional approval of assignments based on percent of correct answers
lista <- list(AnswerKey = list("QuestionId1" = "B",
                               "QuestionId2" = "A"),
              ApproveIfKnownAnswerScoreIsAtLeast = 99)
policya <- do.call(GenerateAssignmentReviewPolicy, lista)

# Example Policy B
## Conditional approval of assignments based on percent of correct answers
## Conditional rejection of assignments based on percent of correct answers
listb <- list(AnswerKey = list("QuestionId1" = "B",
                               "QuestionId2" = "A"),
              ApproveIfKnownAnswerScoreIsAtLeast = 99,
              RejectIfKnownAnswerScoreIsLessThan = 1)
policyb <- do.call(GenerateAssignmentReviewPolicy, listb)

# Example Policy C
## Conditionally extend HIT with more time and assignments based on score
```

```
listc <- list(AnswerKey = list("QuestionId1" = "B"),
              ExtendIfKnownAnswerScoreIsLessThan = 100,
              ExtendMaximumAssignments = 2, # maximum value is 25
              ExtendMinimumTimeInSeconds = seconds(hours = 1))
policyc <- do.call(GenerateAssignmentReviewPolicy, listc)


# Attach an assignment review policy to a HIT
## load template file
## applying Policy B will approve if form answers are "Something" and "Yes"
##                        and reject if both are incorrect
f <- system.file("templates/htmlquestion3.xml", package = "MTurkR")

hit1 <-
CreateHIT(title = "Survey",
          description = "5 question survey",
          reward = ".10",
          expiration = seconds(days = 4),
          duration = seconds(hours = 1),
          keywords = "survey, questionnaire",
          assignment.review.policy = policyb,
          question = GenerateHTMLQuestion(file = f))

# GetReviewResults
GetReviewResultsForHIT(hit1$HITId)

# cleanup
ExpireHIT(hit1$HITId)
DisposeHIT(hit1$HITId)

## End(Not run)
## Not run:
# HIT- and Assignment-level ReviewPolicies

# Example Policy D
## Conditional approval of assignments based on percent of correct answers
listd <- list(AnswerKey = list("QuestionId1" = "B",
                               "QuestionId2" = "A"),
              ApproveIfKnownAnswerScoreIsAtLeast = 99)
policyd <- do.call(GenerateAssignmentReviewPolicy, listd)

# Example Policy E
## Conditionally extend HIT based on HIT Agreement Score
liste <- list(QuestionIds = c("QuestionId3","QuestionId4","QuestionId5"),
              QuestionAgreementThreshold = 65, # at least 2/3rd agreement
              ExtendIfHITAgreementScoreIsLessThan = 66,
              ExtendMinimumTimeInSeconds = 3600,
              ExtendMaximumAssignments = 2,
              DisregardAssignmentIfRejected = TRUE)
policye <- do.call(GenerateHITReviewPolicy, liste)

## load template file
## applying 'policya' will approve if form answers are "Something" and "Yes"
```

```
f <- system.file("templates/htmlquestion3.xml", package = "MTurkR")

# Create HIT
hit2 <-
CreateHIT(title = "Survey",
          description = "5 question survey",
          reward = ".10",
          expiration = seconds(days = 4),
          duration = seconds(hours = 1),
          keywords = "survey, questionnaire",
          assignment.review.policy = policyd,
          hit.review.policy = policye,
          question = GenerateHTMLQuestion(file = f))

# GetReviewResults
GetReviewResultsForHIT(hit2$HITId)

# cleanup
ExpireHIT(hit2$HITId)
DisposeHIT(hit2$HITId)

## End(Not run)
```

`GetAssignment` *Get Assignment(s)*

Description

Get an assignment or multiple assignments for one or more HITs (or a HITType) as a dataframe.

Usage

```
GetAssignment(assignment = NULL, hit = NULL, hit.type = NULL,
              annotation = NULL, status = NULL,
              return.all = FALSE, pagenumber = "1", pagesize = "10",
              sortproperty = "SubmitTime", sortdirection = "Ascending",
              response.group = NULL, return.assignment.dataframe = TRUE,
              verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`assignment`	An optional character string specifying the AssignmentId of an assignment to return. Must specify `assignment` xor `hit` xor `hit.type` xor `annotation`.
`hit`	An optional character string specifying the HITId whose assignments are to be returned, or a vector of character strings specifying multiple HITIds all of whose assignments are to be returned. Must specify `assignment` xor `hit` xor `hit.type` xor `annotation`.

hit.type	An optional character string specifying the HITTypeId (or a vector of HITTypeIds) of one or more HITs whose assignments are to be returned. Must specify assignment xor hit xor hit.type xor annotation.
annotation	An optional character string specifying the value of the RequesterAnnotation field for a batch of HITs. This can be used to retrieve all assignments for all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify assignment xor hit xor hit.type xor annotation.
status	An optional character string (of "Approved","Rejected","Submitted"), specifying whether only a subset of assignments should be returned. If NULL, all assignments are returned (the default). Only applies when hit or hit.type are specified; ignored otherwise.
return.all	If TRUE, all available assignments are returned. Otherwise, only assignments falling within the specified pagenumber and pagesize search results are returned.
pagenumber	An optional character string indicating which page of search results should be returned (only appropriate when specifying a single HITId). Most users can ignore this.
pagesize	An optional character string indicating how many search results should be returned by each request (only appropriate when specifying a single HITId), between 1 and 100. Most users can ignore this.
sortproperty	One of "AcceptTime", "SubmitTime", "AssignmentStatus". Ignored if return.all=TRUE. Most users can ignore this.
sortdirection	Either "Ascending" or "Descending". Ignored if return.all=TRUE. Most users can ignore this.
response.group	An optional character string (or vector of character strings) specifying what details to return. If assignment is specified, response.group can include any of "Request", "Minimal", "AssignmentFeedback", "HITDetail", and/or "HITQuestion". If hit or hit.type is specified, response.group can include "Request", "Minimal", and/or "AssignmentFeedback". For more information, see Common Parameters.
return.assignment.dataframe	A logical specifying whether the Assignment dataframe should be returned. Default is TRUE.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

This function returns the requested assignments. The function must specify an AssignmentId xor a HITId xor a HITTypeId. If an AssignmentId is specified, only that assignment is returned. If a HIT or HITType is specified, default behavior is to return all assignments through a series of sequential (but invisible) API calls meaning that returning large numbers of assignments (or assignments for a large number of HITs in a single request) may be time consuming.

GetAssignments(), assignment(), and assignments() are aliases.

Value

Optionally a dataframe containing Assignment data, including workers responses to any questions specified in the question parameter of the CreateHIT function.

Author(s)

Thomas J. Leeper

References

API Reference: GetAssignment

API Reference: GetAssignmentsForHIT

See Also

GetHIT

ApproveAssignment

ApproveAllAssignments

RejectAssignment

Examples

```
## Not run:
# get an assignment
GetAssignment(assignments="26XXH0JPPSI23H54YVG7BKLEXAMPLE")
# get all assignments for a HIT
GetAssignment(hit="2MQB727M0IGF304GJ16S1F4VE3AYDQ", return.all=TRUE)
# get all assignments for a HITType
GetAssignment(hit.type="2FFNCWYB49F9BBJWA4SJUNST5OFSOW",
              return.all=FALSE, pagenumber="1", pagesize="50")
# get all assignments for an online batch from the RUI
GetAssignment(annotation="BatchId:78382;")

## End(Not run)
```

GetBonuses *Get Bonus Payments*

Description

Get details of bonuses paid to workers, by HIT, HITType, or Assignment.

Usage

```
GetBonuses(assignment = NULL, hit = NULL, hit.type = NULL, annotation = NULL,
           return.all = TRUE, pagenumber = "1", pagesize = "100",
           verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

assignment	An optional character string containing an AssignmentId whose bonuses should be returned. Must specify assignment xor hit xor hit.type xor annotation.
hit	An optional character string containing a HITId whose bonuses should be returned. Must specify assignment xor hit xor hit.type xor annotation.
hit.type	An optional character string containing a HITTypeId (or a vector of HITTypeIds) whose bonuses should be returned. Must specify assignment xor hit xor hit.type xor annotation.
annotation	An optional character string specifying the value of the RequesterAnnotation field for a batch of HITs. This can be used to retrieve bonuses for all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify assignment xor hit xor hit.type xor annotation.
return.all	A logical indicating whether all HITs (as opposed to a specified page of the search results) should be returned. Default is TRUE. Note: This is (temporarily) ignored.
pagenumber	An optional character string indicating which page of search results should be returned. Most users can ignore this.
pagesize	An optional character string indicating how many search results should be returned by each request, between 1 and 100. Most users can ignore this.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

Retrieve bonuses previously paid to a specified HIT, Assignment, or HITType.

bonuses() is an alias.

Value

A dataframe containing the details of each bonus, specifically: AssignmentId, WorkerId, Amount, CurrencyCode, FormattedPrice, Reason, and GrantTime.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GrantBonus

Examples

```
## Not run:
# Get bonuses for a given assignment
GetBonuses(assignment = "26XXH0JPPSI23H54YVG7BKLO82DHNU")

# Get all bonuses for a given HIT
GetBonuses(hit = "2MQB727M0IGF304GJ16S1F4VE3AYDQ")

# Get bonuses from all HITs of a given batch from the RUI
GetBonuses(annotation="BatchId:78382;")

## End(Not run)
```

`GetFileUpload` *Get Files Uploaded by Workers*

Description

Get the URL for a file uploaded by a worker as part of a QuestionForm HIT, or download the file(s) directly to the working directory.

Usage

```
GetFileUpload(assignment, questionIdentifier, download = FALSE,
              open.file.in.browser = FALSE,
              verbose = getOption('MTurkR.verbose', TRUE),
              ...)
```

Arguments

`assignment`	A character string containing an AssignmentId, or a vector of character strings each containing an AssignmentId.
`questionIdentifier`	A question identifer for a file upload question, as specified in the `question` parameter of `CreateHIT` or in the placeholder of a HIT template created in the RUI.
`download`	A logical specifying whether the file(s) should be downloaded and saved in the working directory. Default is `FALSE`.
`open.file.in.browser`	A logical specifying whether the file should be opened in the user's default web browser.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

Note that a FileUploadURL is only valid for 60 seconds (per MTurk documentation), so URLs should either be retrieved one at a time or files should be automatically downloaded to the working directory with the `download = TRUE`. If `browser = TRUE`, request is executed in the user's default web browser, whereas if `open.file.in.browser = TRUE`, the request is executed in R and the file itself is opened in the browser.

If downloaded, files are saved in the local directory with names of the form "QuestionIdentifer_AssignmentId_UploadedFileName.UploadedFileExtension". One may want to use `setwd` to move to an appropriate directory before initiating this operation with `download = TRUE`.

`geturls()` is an alias.

Value

A dataframe containing the AssignmentId, questionIdenifier, temporary file URL, and an indicator of whether each request was valid.

Author(s)

Thomas J. Leeper

References

API Reference

Examples

```
## Not run:
a <- GenerateExternalQuestion("http://www.example.com/","400")
hit1 <-
CreateHIT(title = "Upload a file",
          description = "Upload a file",
          reward = ".10",
          duration = seconds(days=1),
          keywords = "file, upload",
          question = a$string)
ExpireHIT(hit1$HITId)
a <- GetAssignments(hit = hit1$HITId)
f <- GetFileUpload(a, "dictation", download = TRUE)

# cleanup
DisposeHIT(hit1$HITId)

## End(Not run)
```

GetHIT *Get HIT*

Description

Retrieve various details of a HIT as a dataframe. What details are returned depend upon the requested ResponseGroup.

Usage

```
GetHIT(hit, response.group = NULL,
       return.hit.dataframe = TRUE, return.qual.dataframe = TRUE,
       verbose = getOption('MTurkR.verbose', TRUE), ...)

HITStatus(hit = NULL, hit.type = NULL, annotation = NULL,
          verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`hit`	An optional character string specifying the HITId of the HIT to be retrieved. Must specify `hit` xor `hit.type` xor `annotation`.
`hit.type`	An optional character string specifying the HITTypeId (or a vector of HITTypeIds) of the HIT(s) to be retrieved. Must specify `hit` xor `hit.type` xor `annotation`, otherwise all HITs are returned in `HITStatus`.
`annotation`	An optional character string specifying the value of the `RequesterAnnotation` field for a batch of HITs. This can be used to retrieve all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify `hit` xor `hit.type` xor `annotation`, otherwise all HITs are returned in `HITStatus`.
`response.group`	An optional character string (or vector of character strings) specifying what details of each HIT to return of: "Request", "Minimal", "HITDetail", "HITQuestion", "HITAssignmentSummary". For more information, see Common Parameters and HIT Data Structure.
`return.hit.dataframe`	A logical indicating whether the dataframe of HITs should be returned. Default is `TRUE`.
`return.qual.dataframe`	A logical indicating whether the list of each HIT's QualificationRequirements (stored as dataframes in that list) should be returned. Default is `TRUE`.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

GetHIT retrieves characteristics of a HIT. HITStatus is a wrapper that retrieves the Number of Assignments Pending, Number of Assignments Available, Number of Assignments Completed for the HIT(s), which is helpful for checking on the progress of currently available HITs. Specifying a hit.type causes the function to first search for avialable HITs of that HITType, then return the requested information for each HIT.

gethit() and hit() are aliases for GetHIT. status() is an alias for HITStatus.

Value

Optionally a one- or two-element list containing a dataframe of HIT details and, optionally, a list of each HIT's QualificationRequirements (stored as dataframes in that list in the order that HITs were retrieved.).

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GetHITsForQualificationType

GetReviewableHITs

SearchHITs

Examples

```
## Not run:
# register HITType
hittype <-
RegisterHITType(title="10 Question Survey",
                description=
                "Complete a 10-question survey about news coverage and your opinions",
                reward=".20",
                duration=seconds(hours=1),
                keywords="survey, questionnaire, politics")

a <- GenerateExternalQuestion("http://www.example.com/","400")
hit1 <-
CreateHIT(hit.type = hittype$HITTypeId, question = a$string)

GetHIT(hit1$HITId)
HITStatus(hit1$HITId)

# cleanup
DisableHIT(hit1$HITId)
```

```
## End(Not run)
## Not run:
# Get the status of all HITs from a given batch from the RUI
HITStatus(annotation="BatchId:78382;")

## End(Not run)
```

GetHITsForQualificationType
Get HITs by Qualification

Description

Retrieve HITs according to the QualificationTypes that are required to complete those HITs.

Usage

```
GetHITsForQualificationType(qual, response.group = NULL, return.all = TRUE,
                            pagenumber = 1, pagesize = 100,
                            verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual	A character string containing a QualificationTypeId.
response.group	An optional character string specifying what details of each HIT to return of: "Minimal", "Request". For more information, see Common Parameters and HIT Data Structure.
return.all	A logical indicating whether all QualificationTypes (as opposed to a specified page of the search results) should be returned. Default is TRUE.
pagenumber	An optional character string indicating which page of search results should be returned. Most users can ignore this.
pagesize	An optional character string indicating how many search results should be returned by each request, between 1 and 100. Most users can ignore this.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

A function to retrieve HITs that require the specified QualificationType.

gethitsbyqual() is an alias.

Value

A data frame containing the HITId and other requested characteristics of the qualifying HITs.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GetHIT

SearchHITs

Examples

```
## Not run:
q <- ListQualificationTypes()[7,2] # Location requirement
GetHITsForQualificationType(q)

## End(Not run)
```

GetQualificationRequests

Get Qualification Requests

Description

Retrieve workers' requests for a QualificationType.

Usage

```
GetQualificationRequests(qual = NULL, return.all = TRUE,
                         pagenumber = "1", pagesize = "10",
                        sortproperty = "SubmitTime", sortdirection = "Ascending",
                         return.qual.dataframe = TRUE,
                         verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual	An optional character string containing a QualificationTypeId to which the search should be restricted. If none is supplied, requests made for all QualificationTypes are returned.
return.all	A logical indicating whether all QualificationRequestss (as opposed to a specified page of the search results) should be returned. Default is TRUE.
pagenumber	An optional character string indicating which page of search results should be returned. Most users can ignore this.
pagesize	An optional character string indicating how many search results should be returned by each request, between 1 and 100. Most users can ignore this.

sortproperty	Either "SubmitTime" or "QualificationTypeId". Ignored if return.all=TRUE. Most users can ignore this.
sortdirection	Either "Ascending" or "Descending". Ignored if return.all=TRUE. Most users can ignore this.
return.qual.dataframe	A logical indicating whether the QualificationTypes should be returned as a dataframe. Default is TRUE.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

A function to retrieve pending Qualification Requests made by workers, either for a specified QualificationType or all QualificationTypes. Specifically, all active, custom QualificationTypes are visible to workers, and workers can request a QualificationType (e.g., when a HIT requires one they do not have). This function retrieves those requests so that they can be granted (with GrantQualification) or rejected (with RejectQualification).

qualrequests() is an alias.

Value

A dataframe containing the QualificationRequestId, WorkerId, and other information (e.g., Qualification Test results) for each request.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GrantQualification

RejectQualification

Examples

```
## Not run:
GetQualificationRequests()
GetQualificationRequests("2YCIA0RYNJ9262B1D82MPTUEXAMPLE")

## End(Not run)
```

`GetQualifications` *Get Qualifications*

Description

Get all Qualifications of a particular QualificationType assigned to Workers.

Usage

```
GetQualifications(qual, status = NULL, return.all = TRUE,
                  pagenumber = 1, pagesize = 100,
                  verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`qual`	A character string containing a QualificationTypeId for a custom (i.e., not built-in) QualificationType.
`status`	An optional character string specifying whether only "Granted" or "Revoked" Qualifications should be returned.
`return.all`	A logical indicating whether all Qualifications (as opposed to a specified page of the search results) should be returned. Default is `TRUE`.
`pagenumber`	An optional character string indicating which page of search results should be returned. Most users can ignore this.
`pagesize`	An optional character string indicating how many search results should be returned by each request, between 1 and 100. Most users can ignore this.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

A function to retrieve Qualifications granted for the specified QualificationType. To retrieve a specific Qualification score (e.g., for one worker), use `GetQualificationScore`.

A practical use for this is with automatically granted QualificationTypes. After workers request and recieve an automatically granted Qualification that is tied to one or more HITs, `GetQualifications` can be used to retrieve the WorkerId's for workers that are actively working on those HITs (even before they have submitted an assignment).

`getquals()` is an alias.

Value

A dataframe containing the QualificationTypeId, WorkerId, and Qualification scores of workers assigned the Qualification.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GetQualificationScore

UpdateQualificationScore

Examples

```
## Not run:
qual1 <-
AssignQualification(workers = "A1RO9UJNWXMU65",
                    name = "Worked for me before",
                description = "This qualification is for people who have worked for me before",
                    status = "Active",
                    keywords = "Worked for me before")

GetQualifications(qual1$QualificationTypeId)
RevokeQualification(qual1$QualificationTypeId, qual1$WorkerId)
GetQualifications(qual1$QualificationTypeId, status="Revoked")

DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

GetQualificationScore *Get a Worker's Qualification Score*

Description

Get a Worker's score for a specific Qualification. You can only retrieve scores for custom QualificationTypes. Scores for built-in QualificationTypes should be retrieved with GetWorkerStatistic.

Usage

```
GetQualificationScore(qual, workers, verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual	A character string containing a QualificationTypeId for a custom QualificationType.
workers	A character string containing a WorkerId, or a vector of character strings containing multiple WorkerIds, whose Qualification Scores you want to retrieve.

verbose	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
...	Additional arguments passed to `request`.

Details

A function to retrive one or more scores for a specified QualificationType. To retrieve all Qualifications of a given QualificationType, use `GetQualifications` instead. Both `qual` and `workers` can be vectors. If `qual` is not length 1 or the same length as `workers`, an error will occur.

`qualscore()` is an alias.

Value

A dataframe containing the QualificationTypeId, WorkerId, time the qualification was granted, the Qualification score, a column indicating the status of the qualification, and a column indicating whether the API request was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`UpdateQualificationScore`

`GetQualifications`

Examples

```
## Not run:
qual1 <-
AssignQualification(workers = "A1RO9UJNWXMU65",
                    name = "Worked for me before",
                description = "This qualification is for people who have worked for me before",
                    status = "Active",
                    keywords = "Worked for me before")

GetQualificationScore(qual1$QualificationTypeId, qual1$WorkerId)

# cleanup
DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

GetQualificationType *Get QualificationType*

Description

Get the details of a Qualification Type.

Usage

```
GetQualificationType(qual, verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual	A character string containing a QualificationTypeId.
verbose	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
...	Additional arguments passed to `request`.

Details

Retrieve characteristics of a specified QualificationType (as originally specified by `CreateQualificationType`).

`qualtype()` is an alias.

Value

A dataframe containing the QualificationTypeId of the newly created QualificationType and other details as specified in the request.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`CreateQualificationType`

`UpdateQualificationType`

`DisposeQualificationType`

`SearchQualificationTypes`

Examples

```
## Not run:
qual1 <-
CreateQualificationType(name="Worked for me before",
    description="This qualification is for people who have worked for me before",
    status = "Active",
    keywords="Worked for me before")
GetQualificationType(qual1$QualificationTypeId)
DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

`GetReviewableHITs` *Get Reviewable HITs*

Description

Get HITs that are currently reviewable.

Usage

```
GetReviewableHITs(hit.type = NULL, status = NULL, response.group = "Minimal",
                  return.all = TRUE, pagenumber = "1", pagesize = "10",
                  sortproperty = "Enumeration", sortdirection = "Ascending",
                  verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`hit.type`	An optional character string containing a HITTypeId to consider when looking for reviewable HITs.
`status`	An optional character string of either "Reviewable" or "Reviewing" limiting the search to HITs of with either status.
`response.group`	A character string specifying what details of each HIT to return. API currently only supports "Minimal". For more information, see Common Parameters and HIT Data Structure.
`return.all`	A logical indicating whether all QualificationTypes (as opposed to a specified page of the search results) should be returned. Default is `TRUE`.
`pagenumber`	An optional character string indicating which page of search results should be returned. Most users can ignore this.
`pagesize`	An optional character string indicating how many search results should be returned by each request, between 1 and 100. Most users can ignore this.
`sortproperty`	One of "Title", "Reward", "Expiration", "CreationTime", "Enumeration". Ignored if `return.all=TRUE`. Most users can ignore this.
`sortdirection`	Either "Ascending" or "Descending". Ignored if `return.all=TRUE`. Most users can ignore this.

verbose	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
...	Additional arguments passed to `request`.

Details

A simple function to return the HITIds of HITs currently in "Reviewable" or "Reviewing" status. To retrieve additional details about each of these HITs, see `GetHIT`. This is an alternative to `SearchHITs`.

`reviewable()` is an alias.

Value

A dataframe containing only a column of HITIds.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`GetHIT`

`GetHITsForQualificationType`

`SearchHITs`

Examples

```
## Not run:
GetReviewableHITs()

## End(Not run)
```

`GetReviewResultsForHIT`

Get ReviewPolicy Results for a HIT

Description

Get HIT- and/or Assignment-level ReviewPolicy Results for a HIT

Usage

```
GetReviewResultsForHIT(hit, assignment = NULL, policy.level = NULL,
                       retrieve.results = TRUE, retrieve.actions = TRUE,
                       return.all = FALSE, pagenumber = 1, pagesize = 400,
                       verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

hit	A character string containing a HITId.
assignment	An optional character string containing an AssignmentId. If specified, only results pertaining to that assignment will be returned.
policy.level	Either HIT or Assignment. If NULL (the default), all data for both policy levels is retrieved.
retrieve.results	Optionally retrieve ReviewResults. Default is TRUE.
retrieve.actions	Optionally retrieve ReviewActions. Default is TRUE.
return.all	A logical indicating whether all Qualifications (as opposed to a specified page of the search results) should be returned. Default is FALSE.
pagenumber	An optional character string indicating which page of search results should be returned. Most users can ignore this.
pagesize	An optional character string indicating how many search results should be returned by each request, between 1 and 400. Most users can ignore this.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

A simple function to return the results of a ReviewPolicy. This is intended only for advanced users, who should reference MTurk documentation for further information or see the notes in GenerateHITReviewPolicy.

reviewresults is an alias.

Value

A four-element list containing up to four named dataframes, depending on what ReviewPolicy (or ReviewPolicies) were attached to the HIT and whether results or actions are requested: AssignmentReviewResult, AssignmentReviewAction, HITReviewResult, and/or HITReviewAction.

Author(s)

Thomas J. Leeper

References

API Reference

API Reference (ReviewPolicies)

API Reference (Data Structure)

See Also

CreateHIT

GenerateHITReviewPolicy

GetStatistic *MTurk Worker and Requester Statistics*

Description

Get a requester statistic or a statistic for a particular worker. RequesterReport and WorkerReport provide wrappers that return all available statistics.

Usage

```
GetStatistic(statistic, period = "LifeToDate", count = NULL,
             response.group = NULL,
             verbose = getOption('MTurkR.verbose', TRUE), ...)
RequesterReport(period = "LifeToDate",
                verbose = getOption('MTurkR.verbose', TRUE), ...)

GetWorkerStatistic(worker, statistic, period = "LifeToDate", count = NULL,
                   response.group = NULL,
                   verbose = getOption('MTurkR.verbose', TRUE), ...)
WorkerReport(worker, period = "LifeToDate",
             verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

worker	A character string containing a WorkerId.
statistic	A character string containing the name of a statistic. Statistics can be retrieved from ListStatistics.
period	One of: "OneDay", "SevenDays", "ThirtyDays", "LifeToDate". Default is "LifeToDate".
count	If period="OneDay", the number of days to return. Default is 1 (the most recent day).
response.group	An optional character string (or vector of character strings) specifying what details to return of "Request", "Minimal", or "Parameters". For more information, see Common Parameters.

verbose	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
...	Additional arguments passed to `request`.

Details

Retrieve a specific requester or worker statistic. The list of available statistics can be retrieved by calling `ListStatistics`. Useful for monitoring workers or one's own use of the requester system.

`statistic()` is an alias for `GetStatistic`. `workerstatistic()` is an alias for `GetWorkerStatistic`.

Value

A dataframe containing the Date, Value, and Statistic (and WorkerId, if `GetWorkerStatistic` or `WorkerReport` are called), and the value thereof. `GetStatistic` and `GetWorkerStatistic` return only information about the requested statistic as a data.frame. `RequesterReport` and `WorkerReport` return all of the requester and worker statistics, respectively, that are available in `ListStatistics`.

Author(s)

Thomas J. Leeper

References

API Reference: Requester Statistics

API Reference: Worker Statistics

See Also

`ListStatistics`

Examples

```
## Not run:
GetStatistic("NumberHITsSubmitted","OneDay")
RequesterReport("ThirtyDays")
GetWorkerStatistic("A1RO9UJNWXMU65","PercentHITsApproved","LifeToDate")
WorkerReport("A1RO9UJNWXMU65","SevenDays")

## End(Not run)
```

GrantBonus *Pay Bonus to Worker*

Description

Pay a bonus to one or more workers. This function spends money from your MTurk account and will fail if insufficient funds are available.

Usage

```
GrantBonus(workers, assignments, amounts, reasons,
           unique.request.token = NULL,
           verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

workers	A character string containing a WorkerId, or a vector of character strings containing multiple WorkerIds.
assignments	A character string containing an AssignmentId for an assignment performed by that worker, or a vector of character strings containing the AssignmentId for an assignment performed by each of the workers specified in workers.
amounts	A character string containing an amount (in U.S. Dollars) to bonus the worker(s), or a vector (of length equal to the number of workers) of character strings containing the amount to be paid to each worker.
reasons	A character string containing a reason for bonusing the worker(s), or a vector (of length equal to the number of workers) of character strings containing the reason to bonus each worker. The reason is visible to each worker.
unique.request.token	An optional character string, included only for advanced users. It can be used to prevent resending a bonus. A bonus will not be granted if a bonus was previously granted (within a short time window) using the same unique.request.token.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

A simple function to grant a bonus to one or more workers. The function is somewhat picky in that it requires a WorkerId, the AssignmentId for an assignment that worker has completed, an amount, and a reason for the bonus, for each bonus to be paid. Optionally, the amount and reason can be specified as single (character string) values, which will be used for each bonus.

bonus() and paybonus() are aliases.

Value

A dataframe containing the WorkerId, AssignmentId, amount, reason, and whether each request to bonus was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GetBonuses

Examples

```
## Not run:
# Grant a single bonus
a <- "A1RO9UEXAMPLE"
b <- "26XXH0JPPSI23H54YVG7BKLEXAMPLE"
c <- ".50"
d <- "Thanks for your great work on my HITs!"
GrantBonus(workers=a, assignments=b, amounts=c, reasons=d)

## End(Not run)
## Not run:
# Grant bonuses to multiple workers
a <- c("A1RO9EXAMPLE1","A1RO9EXAMPLE2","A1RO9EXAMPLE3")
b <-
c("26XXH0JPPSI23H54YVG7BKLEXAMPLE1",
"26XXH0JPPSI23H54YVG7BKLEXAMPLE2",
"26XXH0JPPSI23H54YVG7BKLEXAMPLE3")
c <- c(".50",".10",".25")
d <- "Thanks for your great work on my HITs!"
GrantBonus(workers=a, assignments=b, amounts=c, reasons=d)

## End(Not run)
```

GrantQualification *Grant/Reject Qualification Request*

Description

Grant or reject a worker's request for a Qualification.

Usage

```
GrantQualification(qual.requests, values,
                   verbose = getOption('MTurkR.verbose', TRUE), ...)

RejectQualification(qual.requests, reason = NULL,
                    verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual.requests	A character string containing a QualificationRequestId (for example, returned by GetQualificationRequests), or a vector of QualificationRequestIds.
values	A character string containing the value of the Qualification to be assigned to the worker, or a vector of values of length equal to the number of QualificationRequests.
reason	An optional character string, or vector of character strings of length equal to length of the qual.requests parameter, supplying each worker with a reason for rejecting their request for the Qualification. Workers will see this message. Maximum of 1024 characters.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

Qualifications are publicly visible to workers on the MTurk website and workers can request Qualifications (e.g., when a HIT requires a QualificationType that they have not been assigned). QualificationRequests can be retrieved via GetQualificationRequests. GrantQualification grants the specified qualification requests. Requests can be rejected with RejectQualifications.

Note that granting a qualification may have the consequence of modifying a worker's existing qualification score. For example, if a worker already has a score of 100 on a given QualificationType and then requests the same QualificationType, a GrantQualification action might increase or decrease that worker's qualification score.

Similarly, rejecting a qualification is not the same as revoking a worker's Qualification. For example, if a worker already has a score of 100 on a given QualificationType and then requests the same QualificationType, a RejectQualification leaves the worker's existing Qualification in place. Use RevokeQualification to entirely remove a worker's Qualification.

GrantQualifications() and grantqual() are aliases; RejectQualifications() and rejectrequest() are aliases.

Value

A dataframe containing the QualificationRequestId, reason for rejection (if applicable; only for RejectQualification), and whether each request was valid.

Author(s)

Thomas J. Leeper

References

API Reference: GrantQualification

API Reference: RejectQualification

See Also

GetQualificationRequests

Examples

```
## Not run:
# create QualificationType
qual1 <- CreateQualificationType(name="Requestable Qualification",
           description="This is a test qualification that can be requested.",
           status = "Active")

# poll for qualification requests
qrs <- GetQualificationRequests(qual1$QualificationTypeId)

# grant a qualification request
GrantQualification(qrs$QualificationRequestId[1], values = "100")

# correct a worker's score (note use of `SubjectId`, not `WorkerId`)
UpdateQualificationScore(qrs$QualificationTypeId[1], qrs$SubjectId[1], value = "95")

# reject a qualification request
RejectQualification(qrs$QualificationTypeId[2], reason = "Sorry!")

# cleanup
DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

Miscellaneous *Convenience Functions and RUI Interaction*

Description

These functions provide lists (of QualificationTypes, Requester and Worker Statistics) useful when creating HITs and QualificationTypes, and other functions to open specific pages of the MTurk Requester User Interface (RUI), or in the case of ViewAvailableHITs, the MTurk worker site.

Usage

```
ListOperations(op = NULL)

ListQualificationTypes(qual = NULL)

ListStatistics(stat = NULL, value.type = NULL, type = NULL)

ViewAvailableHITs(query = NULL, requester = NULL, min.reward = NULL, qualified = NULL)

OpenWorkerPage(workerid = NULL)
OpenManageHITPage(hit = NULL)
```

```
OpenDownloadPage(hit, download = FALSE)
OpenQualificationPage()
```

Arguments

op	For ListOperations: a number indicating which of the operations to return. Probably not useful.
qual	For ListQualificationTypes: a character string containing the name of a built-in QualificationType. If specified, ListQualificationTypes returns only the QualificationTypeId of that QualificationType.
stat	For ListStatistics: an optional character string specifying the name of an MTurk statistic. If specified, only details of that statistic are returned.
value.type	For ListStatistics: an optional character string specifying whether "Double"-type or "Long"-type statistics should be returned. If NULL, both types are returned. Probably not useful.
type	For ListStatistics: an optional character string specifying whether "GetRequesterStatistic"-type or "GetRequesterWorkerStatistic"-type statistics should be returned. If NULL, both types are returned.
query	For ViewAvailableHITs: an optional character string containing a search query used to search through HITs available on the MTurk worker site.
requester	For ViewAvailableHITs: an optional character string containing a RequesterId whose HITs are to be searched, for example to see how one's own HITs appear to workers on the MTurk worker site.
min.reward	For ViewAvailableHITs: an optional character string containing a minimum reward (in U.S. Dollars) criterion to be used when searching available HITs on the worker site.
qualified	For ViewAvailableHITs: an optional logical specifying whether only HITs for which you are qualified should be searched.
workerid	For OpenWorkerPage: an optional character string containing the WorkerId of a worker whose Requester User Interface (RUI) management page is to be opened. If NULL, function opens the list of workers.
hit	For OpenManageHITPage: an optional character string containing the HITId of a HIT whose Requester User Interface (RUI) management page is to be opened. If NULL, function opens the list of all HITs. For OpenDownloadPage: a mandatory character string containing a HITID of a HIT whose assignment data download page should be opened.
download	For OpenDownloadPage: a logical indicating whether the HIT results should be downloaded directly or whether the Requester User Interface (RUI) ManageHIT page should be opened (the default).

Details

A set of convenience functions to either display various information about MTurk semantics or open specified parts of the Requester User Interface (RUI).

Value

Either a dataframe containing the requested information (in the case of `ListOperations`, `ListQualificationTypes`, `ListStatistics`) or nothing internal to R, but the specified webpage is opened in the user's default web browser (in the case of `OpenWorkerPage`, `OpenManageHITPage`, `OpenDownloadPage`, or `OpenQualificationPage`).

Author(s)

Thomas J. Leeper

Examples

```
## Not run:
ListOperations()
ListQualificationTypes()
ListStatistics()
ViewAvailableHITs(min.reward=".50")

OpenWorkerPage()
OpenWorkerPage("A1RO9UEXAMPLE")
OpenManageHITPage()
OpenDownloadPage("267SI2KYEPLS8QWZYPXWP3EXAMPLE",download=TRUE)
OpenQualificationPage()

## End(Not run)
```

`readlogfile` *Read the MTurkR Logfile*

Description

A log of all MTurk API requests are stored in a tab-separated value file called 'MTurkRLog.tsv' in, by default, the current working directory. This function reads this MTurkR logfile into R as a dataframe, using `read.delim`.

Usage

```
readlogfile(path = getOption('MTurkR.logdir'), filename="MTurkRlog.tsv")
```

Arguments

`path`	An optional character string specifying the path of a directory containing an MTurkR log file.
`filename`	The name of the MTurkR log file. The default is '`MTurkRlog.tsv`'.

Details

By default, MTurkR stores a record of all MTurk API requests in a local file in the working directory (though this can be reset with `options('MTurkR.logdir')`. This function reads the locally stored MTurkR log file ('`MTurkRlog.tsv`') into R as a dataframe. This is useful for error checking and reviewing prior requests.

Value

A dataframe containing details of previous (logged) MTurk API requests (including RequestId, Operation performed, REST query parameters, and MTurk response XML as a character string).

Author(s)

Thomas J. Leeper

See Also

`request`

Examples

```
## Not run:
log <- readlogfile()

## End(Not run)
```

`RegisterHITType` *Register a HITType*

Description

Register a HITType on MTurk, in order to create one or more HITs to show up as a group to workers.

Usage

```
RegisterHITType(title, description, reward, duration, keywords = NULL,
                auto.approval.delay = NULL, qual.req = NULL,
                verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`title`	A character string containing the title for the HITType. All HITs of this HITType will be visibly grouped to workers according to this title. Maximum of 128 characters.
`description`	A character string containing a description of the HITType. This is visible to workers. Maximum of 2000 characters.

reward	A character string containing the per-assignment reward amount, in U.S. Dollars (e.g., "0.15").
duration	A character string containing the amount of time workers have to complete an assignment for HITs of this HITType, in seconds (for example, as returned by seconds). Minimum of 30 seconds and maximum of 365 days.
keywords	An optional character string containing a comma-separated set of keywords by which workers can search for HITs of this HITType. Maximum of 1000 characters.
auto.approval.delay	An optional character string specifying the amount of time, in seconds (for example, as returned by seconds), before a submitted assignment is automatically granted. Maximum of 30 days.
qual.req	An optional character string containing one or more QualificationRequirements data structures, for example as returned by GenerateQualificationRequirement.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

All HITs of a given HITType are visibly grouped together for workers and share common properties (e.g., reward amount, QualificationRequirements). This function registers a HITType in the MTurk system, which can then be used when creating individual HITs. If a requester wants to change these properties for a specific HIT, the HIT should be changed to a new HITType (see ChangeHITType).

hittype() is an alias.

Value

A two-column dataframe containing the HITTypeId of the newly registered HITType and an indicator for whether the registration request was valid.

Author(s)

Thomas J. Leeper

References

API Reference: Operation

API Reference: Concept

See Also

CreateHIT

ChangeHITType

Examples

```
## Not run:
RegisterHITType(title="10 Question Survey",
                description=
                "Complete a 10-question survey about news coverage and your opinions",
                reward=".20",
                duration=seconds(hours=1),
                keywords="survey, questionnaire, politics")

## End(Not run)
```

`RejectAssignment` *Reject Assignment*

Description

Reject a Worker's assignment (or multiple assignments) submitted for a HIT. Feedback should be provided for why an assignment was rejected.

Usage

```
RejectAssignment(assignments, feedback = NULL,
                 verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`assignments`	A character string containing an AssignmentId, or a vector of multiple character strings containing multiple AssignmentIds, to reject.
`feedback`	An optional character string containing any feedback for a worker. This must have length 1 or length equal to the number of workers.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

Reject assignments, by AssignmentId (as returned by `GetAssignment`). More advanced functionality to quickly reject many or all assignments (ala `ApproveAllAssignments`) is intentionally not provided.

`RejectAssignments()` and `reject()` are aliases.

Value

A dataframe containing the list of AssignmentIds, feedback (if any), and whether or not each rejection request was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

ApproveAssignment

Examples

```
## Not run:
RejectAssignment(assignments="26XXH0JPPSI23H54YVG7BKLEXAMPLE")

## End(Not run)
```

---

request *Execute an MTurk API Request*

Description

This is the workhorse function that makes authenticated HTTP requests to the MTurk API. It is not exported as of v0.5.

Usage

```
request(operation, GETparameters = NULL,
        keypair = c(Sys.getenv("AWS_ACCESS_KEY_ID"),
                    Sys.getenv("AWS_SECRET_ACCESS_KEY")),
        browser = getOption('MTurkR.browser', FALSE),
        log.requests = getOption('MTurkR.log', TRUE),
        sandbox = getOption('MTurkR.sandbox', FALSE),
        verbose = getOption('MTurkR.verbose', TRUE),
        validation.test = getOption('MTurkR.test', FALSE),
        service = "AWSMechanicalTurkRequester",
        version = NULL)
```

Arguments

operation	The MTurk API operation to be performed.
GETparameters	An optional character string containing URL query parameters that specify options for the request.
keypair	A two-element character vector containing an AWS Access Key ID and an AWS Secret Access Key. The easiest way to do this is to specify 'AWS_ACCESS_KEY_ID' and 'AWS_SECRET_ACCESS_KEY' environment variables using Sys.setenv() or by placing these values in an .Renviron file.

browser	Optionally open the request in the default web browser, rather than opening in R. Default is FALSE.
log.requests	A logical specifying whether API requests should be logged. Default is TRUE. See readlogfile for details. The log is stored, by default, in the working directory at the time MTurkR is loaded. This can be changed using options("MTurkR.logdir"); if not set, the current working directory is used.
sandbox	Optionally execute the request in the MTurk sandbox rather than the live server. Default is FALSE.
verbose	Whether errors produced by the MTurk API request should be printed.
validation.test	Currently a logical that causes request to return the URL of the specified REST request. Default is FALSE. May additionally validate the request (and supply information about that validation) in the future.
service	The MTurk service to which the authenticated request will be sent. Supplied only for advanced users.
version	The version of the MTurk API to use.

Details

This is an internal function that executes MTurk API requests. It is made available for use by advanced users to execute custom API requests.

Value

A list of class "MTurkResponse" containing:

operation	A character string identifying the MTurk API operation performed.
request.id	The Request ID created by the API request.
request.url	The URL of the MTurk API REST request.
valid	A logical indicating whether or not the request was valid and thus executed as intended.
xml	A character string containing the XML API response.

Author(s)

Thomas J. Leeper

References

API Reference

RevokeQualification *Revoke a Qualification from a Worker*

Description

Revoke a Qualification from a worker or multiple workers. This deletes their qualification score and any record thereof.

Usage

```
RevokeQualification(qual, worker, reason = NULL,
                    verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual	A character string containing a QualificationTypeId.
worker	A character string containing a WorkerId, or a vector of character strings containing multiple WorkerIds.
reason	An optional character string, or vector of character strings of length equal to length of the workers parameter, supplying each worker with a reason for revoking their Qualification. Workers will see this message.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

A simple function to revoke a Qualification assigned to one or more workers.

RevokeQualifications() and revokequal() are aliases.

Value

A dataframe containing the QualificationTypeId, WorkerId, reason (if applicable), and whether each request was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GrantQualification

RejectQualification

Examples

```
## Not run:
qual1 <-
AssignQualification(workers = "A1RO9UJNWXMU65",
                    name = "Worked for me before",
                description = "This qualification is for people who have worked for me before",
                    status = "Active",
                    keywords = "Worked for me before")

RevokeQualification(qual = qual1$QualificationTypeId,
                    worker = qual1$WorkerId,
                    reason = "No longer needed")

DisposeQualificationType(qual1$QualificationTypeId)


## End(Not run)
```

`SearchHITs` *Search your HITs*

Description

Search for your HITs and return those HITs as R objects.

Usage

```
SearchHITs(response.group = NULL, return.all = TRUE,
           pagenumber = "1", pagesize = "10",
           sortproperty = "Enumeration", sortdirection = "Ascending",
           return.hit.dataframe = TRUE, return.qual.dataframe = TRUE,
           verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`response.group`	An optional character string (or vector of character strings) specifying what details of each HIT to return of: "Request", "Minimal", "HITDetail", "HITQuestion", "HITAssignmentSummary". For more information, see Common Parameters and HIT Data Structure.
`return.all`	A logical indicating whether all HITs (as opposed to a specified page of the search results) should be returned. Default is `TRUE`.
`pagenumber`	An optional character string indicating which page of search results should be returned. Most users can ignore this.
`pagesize`	An optional character string indicating how many search results should be returned by each request, between 1 and 100. Most users can ignore this.
`sortproperty`	One of "Title", "Reward", "Expiration", "CreationTime", "Enumeration". Ignored if `return.all=TRUE`. Most users can ignore this.

`sortdirection` Either "Ascending" or "Descending". Ignored if `return.all=TRUE`. Most users can ignore this.

`return.hit.dataframe`
A logical indicating whether the dataframe of HITs should be returned. Default is `TRUE`.

`return.qual.dataframe`
A logical indicating whether the list of each HIT's QualificationRequirements (stored as dataframes in that list) should be returned. Default is `TRUE`.

`verbose` Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.

`...` Additional arguments passed to `request`.

Details

Retrieve your current HITs (and, optionally, characteristics thereof). To view HITs on the MTurk requester website, see `OpenManageHITPage`. To view HITs on the MTurk worker website, use `ViewAvailableHITs`.

`searchhits()` is an alias.

Value

A list one- or two-element list containing a dataframe of HIT details and, optionally (if 'return.qual.dataframe = TRUE'), a list of each HIT's QualificationRequirements (stored as dataframes in that list).

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`GetHIT`

`GetReviewableHITs`

`SearchQualificationTypes`

`ViewAvailableHITs`

Examples

```
## Not run:
SearchHITs()

## End(Not run)
```

SearchQualificationTypes
Search QualificationTypes

Description

Search for available QualificationTypes, including yours and others available on the MTurk system created by other requesters.

Usage

```
SearchQualificationTypes(query = NULL, only.mine = TRUE, only.requestable = FALSE,
                         return.all = FALSE, pagenumber = "1", pagesize = "10",
                         sortproperty = "Name", sortdirection = "Ascending",
                         return.qual.dataframe = TRUE,
                         verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

query	An optional character string containing a search query to be used to search among available QualificationTypes.
only.mine	A logical indicating whether only your QualificationTypes should be returned (the default). If FALSE, QualificationTypes created by all requesters will be returned.
only.requestable	A logical indicating whether only requestable QualificationTypes should be returned. Default is FALSE.
return.all	A logical indicating whether all QualificationTypes (as opposed to a specified page of the search results) should be returned. Default is TRUE.
pagenumber	An optional character string indicating which page of search results should be returned. Most users can ignore this.
pagesize	An optional character string indicating how many search results should be returned by each request, between 1 and 100. Most users can ignore this.
sortproperty	API currently only supports "Name". Most users can ignore this.
sortdirection	Either "Ascending" or "Descending". Ignored if return.all=TRUE. Most users can ignore this.
return.qual.dataframe	A logical indicating whether the QualificationTypes should be returned as a dataframe. Default is TRUE.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

Retrieve available QualificationTypes, optionally only those QualificationTypes created by you and/or those that meet specific search criteria specified in the `query` parameter. Given that the total number of QualificationTypes available from all requesters could be infinitely large, specifying both `only.mine=FALSE` and `return.all=FALSE` will be time-consuming and may cause memory problems.

`searchquals()` is an alias.

Value

A dataframe containing the QualificationTypeId of the newly created QualificationType and other details as specified in the request.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

`GetQualificationType`

`CreateQualificationType`

`UpdateQualificationType`

`DisposeQualificationType`

`SearchHITs`

Examples

```
## Not run:
SearchQualificationTypes(only.mine = TRUE,
                         return.all = TRUE)
SearchQualificationTypes(query = "MIT",
                         only.mine = FALSE,
                         return.all = FALSE)

## End(Not run)
```

seconds *Convert arbitrary times to seconds*

Description

A convenience function to convert arbitrary numbers of days, hours, minutes, and/or seconds into seconds.

Usage

```
seconds(days = NULL, hours = NULL, minutes = NULL, seconds = NULL)
```

Arguments

days	An optional number of days.
hours	An optional number of hours.
minutes	An optional number of minutes.
seconds	An optional number of seconds.

Details

A convenience function to convert arbitrary numbers of days, hours, minutes, and/or seconds into seconds. For example, to be used in setting a HIT expiration time. MTurk only accepts times (e.g., for HIT expirations, or the duration of assignments) in seconds. This function returns an integer value equal to the number of seconds of the input, and can be used atomically within other MTurkR calls (e.g., CreateHIT).

Value

An integer equal to the requested amount of time in seconds.

Author(s)

Thomas J. Leeper

Examples

```
seconds(hours=5, seconds=15)
seconds(days=1)
```

SendTestEventNotification *Test a Notification*

Description

Test a HITType Notification, for example, to try out a HITType Notification before creating a HIT.

Usage

```
SendTestEventNotification(notification, test.event.type,
                          verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

notification	A character string containing a URL query parameter-formatted Notification structure (e.g., returned by GenerateNotification).
test.event.type	A character string containing one of: AssignmentAccepted, AssignmentAbandoned, AssignmentReturned, AssignmentSubmitted, HITReviewable, HITExpired (the default), or Ping.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

Test a Notification configuration. The test mimics whatever the Notification configuration will do when the event described in test.event.type occurs. For example, if a Notification has been configured to send an email any time an Assignment is Submitted, testing for an AssignmentSubmitted event should trigger an email. Similarly, testing for an AssignmentReturned event should do nothing.

notificationtest is an alias.

Value

A dataframe containing the notification, the event type, and details on whether the request was valid. As a side effect, a notification will be sent to the configured destination (either an email or an SQS queue).

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GenerateNotification

SetHITTypeNotification

Examples

```
## Not run:
hittype <-
RegisterHITType(title="10 Question Survey",
                description=
                "Complete a 10-question survey about news coverage and your opinions",
                reward=".20",
                duration=seconds(hours=1),
                keywords="survey, questionnaire, politics")

a <- GenerateNotification("requester@example.com", event.type = "HITExpired")
SetHITTypeNotification(hit.type = hittype$HITTypeId,
                       notification = a,
                       active = TRUE)
# send test notification
SendTestEventNotification(a, test.event.type="HITExpired")

## End(Not run)
```

SetHITAsReviewing *Set HIT as "Reviewing"*

Description

Update the review status of a HIT, from "Reviewable" to "Reviewing" or the reverse.

Usage

```
SetHITAsReviewing(hit = NULL, hit.type = NULL, annotation = NULL,
                  revert = FALSE,
                  verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

hit	An optional chracter character string containing a HITId, or a vector character strings containing HITIds, whose status are to be changed. Must specify hit xor hit.type xor annotation.
hit.type	An optional character string specifying a HITTypeId (or a vector of HITTypeIds), all the HITs of which should be set as "Reviewing" (or the reverse). Must specify hit xor hit.type xor annotation.

annotation	An optional character string specifying the value of the RequesterAnnotation field for a batch of HITs. This can be used to set the review status all HITs from a "batch" created in the online Requester User Interface (RUI). To use a batch ID, the batch must be written in a character string of the form "BatchId:78382;", where "73832" is the batch ID shown in the RUI. Must specify hit xor hit.type xor annotation.
revert	An optional logical to revert the HIT from "Reviewing" to "Reviewable".
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

A function to change the status of one or more HITs (or all HITs of a given HITType) to "Reviewing" or the reverse. This affects what HITs are returned by GetReviewableHITs. Must specify a HITId xor a HITTypeId.

reviewing() is an alias.

Value

A dataframe containing HITId, status, and whether the request to change the status of each was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GetReviewableHITs

Examples

```
## Not run:
a <- GenerateExternalQuestion("http://www.example.com/","400")
hit1 <-
CreateHIT(hit.type="2FFNCWYB49F9BBJWA4SJUNST5OFSOW", question=a$string)
SetHITAsReviewing(hit1$HITId)

# cleanup
DisableHIT(hit1$HITId)

## End(Not run)
```

SetHITTypeNotification

Configure a HITType Notification

Description

Configure a notification to be sent when specific actions occur for the specified HITType.

Usage

```
SetHITTypeNotification(hit.type, notification = NULL, active = NULL,
                       verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

hit.type	A character string specifying the HITTypeId of the HITType for which notifications are being configured.
notification	A character string containing a URL query parameter-formatted Notification structure (e.g., returned by GenerateNotification).
active	A logical indicating whether the Notification is active or inactive.
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

Configure a notification to be sent to the requester whenever an event (specified in the Notification object) occurs. This is useful, for example, to enable email notifications about when assignments are submitted or HITs are completed, or for other HIT-related events.

Email notifications are useful for small projects, but configuring notifications to use the Amazon Simple Queue Service (SQS) is more reliable for large projects and allows automated processing of notifications. See examples for SQS examples. Note that using SQS requires an SQS queue with permissions configured to allow SendMessage requests from MTurk (the "principal" for MTurk is arn:aws:iam::755651556756:user/MTurk-SQS). See the MTurkR wiki for an extended tutorial.

setnotification() is an alias.

Value

A dataframe containing details of the Notification and whether or not the request was successfully executed by MTurk.

Once configured, events will trigger a side effect in the form of a notification sent to the specified transport (either an email address or SQS queue). That notification will contain the following details: EventType, EventTime, HITTypeId, HITId, and (if applicable) AssignmentId.

Author(s)

Thomas J. Leeper

References

API Reference: Operation

API Reference: Concept

See Also

GenerateNotification

SendTestEventNotification

Examples

```
## Not run:
# setup email notification
hittype <-
RegisterHITType(title="10 Question Survey",
                description=
                "Complete a 10-question survey about news coverage and your opinions",
                reward=".20",
                duration=seconds(hours=1),
                keywords="survey, questionnaire, politics")

a <- GenerateNotification("requester@example.com", event.type = "HITExpired")
SetHITTypeNotification(hit.type = hittype$HITTypeId,
                       notification = a,
                       active = TRUE)
# send test notification
SendTestEventNotification(a, test.event.type="HITExpired")

## End(Not run)

## Not run:
# setup SQS notification
hittype <-
RegisterHITType(title="10 Question Survey",
                description=
                "Complete a 10-question survey about news coverage and your opinions",
                reward=".20",
                duration=seconds(hours=1),
                keywords="survey, questionnaire, politics")

b <- GenerateNotification("https://sqs.us-east-1.amazonaws.com/123456789/Example",
                          transport = "SQS",
                          event.type = "HITExpired")
SetHITTypeNotification(hit.type = hittype$HITTypeId,
                       notification = b,
                       active = TRUE)
# send test notification
```

```
SendTestEventNotification(b, test.event.type="HITExpired")

## End(Not run)
```

`UpdateQualificationScore` *Update a worker's score for a QualificationType*

Description

Update a worker's score for a QualificationType that you created. Scores for built-in QualificationTypes (e.g., location, worker statistics) cannot be updated.

Usage

```
UpdateQualificationScore(qual, workers, values = NULL, increment = NULL,
                         verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

`qual`	A character string containing a QualificationTypeId.
`workers`	A character string containing a WorkerId, or a vector of character strings containing multiple WorkerIds.
`values`	A character string containing an integer value to be assigned to the worker, or a vector of character strings containing integer values to be assigned to each worker (and thus must have length equal to the number of workers).
`increment`	An optional character string specifying, in lieu of "values", the amount that each worker's current QualfiicationScore should be increased.
`verbose`	Optionally print the results of the API request to the standard output. Default is taken from `getOption('MTurkR.verbose', TRUE)`.
`...`	Additional arguments passed to `request`.

Details

A function to update the Qualification score assigned to one or more workers for the specified custom QualificationType. The simplest use is to specify a QualificationTypeId, a WorkerId, and a value to be assigned to the worker. Scores for multiple workers can be updated in one request.

Additionally, the `increment` parameter allows you to increase (or decrease) each of the specified workers scores by the specified amount. This might be useful, for example, to keep a QualificationType that records how many of a specific style of HIT a worker has completed and increase the value of each worker's score by 1 after they complete a HIT.

`updatequalscore()` is an alias.

Value

A dataframe containing the QualificationTypeId, WorkerId, Qualification score, and whether the request to update each was valid.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GetQualificationScore

GetQualifications

Examples

```
## Not run:
qual1 <-
CreateQualificationType(name="Worked for me before",
    description="This qualification is for people who have worked for me before",
    status = "Active",
    keywords="Worked for me before")
AssignQualification(qual1$QualificationTypeId, "A1RO9UJNWXMU65", value="50")
UpdateQualificationScore(qual1$QualificationTypeId, "A1RO9UJNWXMU65", value="95")
UpdateQualificationScore(qual1$QualificationTypeId, "A1RO9UJNWXMU65", increment="1")
DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

UpdateQualificationType

Update a Worker QualificationType

Description

Update characteristics of a QualificationType.

Usage

```
UpdateQualificationType(qual, description = NULL, status = NULL,
                        retry.delay = NULL, test = NULL, answerkey = NULL,
                        test.duration = NULL,
                        validate.test = FALSE, validate.answerkey = FALSE,
                        auto = NULL, auto.value = NULL,
                        verbose = getOption('MTurkR.verbose', TRUE), ...)
```

Arguments

qual	A character string containing a QualificationTypeId.
description	A character string describing the Qualification. This is visible to workers. Maximum of 2000 characters.
status	A character vector of "Active" or "Inactive", indicating whether the QualificationType should be active and visible.
retry.delay	An optional time (in seconds) indicating how long workers have to wait before requesting the QualificationType after an initial rejection.
test	An optional character string consisting of a QuestionForm data structure, used as a test a worker must complete before the QualificationType is granted to them.
answerkey	An optional character string consisting of an AnswerKey data structure, used to automatically score the test. If a previous test with an associated AnswerKey is updated, the new test will not have an AnswerKey unless a new one is included in the same call (even if it is identical to the previous AnswerKey).
test.duration	An optional time (in seconds) indicating how long workers have to complete the test.
validate.test	A logical specifying whether the test parameter should be validated against the relevant MTurk schema prior to creating the QualificationType (operation will fail if it does not validate, and will return validation information). Default is FALSE.
validate.answerkey	A logical specifying whether the answerkey parameter should be validated against the relevant MTurk schema prior to creating the QualificationType (operation will fail if it does not validate, and will return validation information). Default is FALSE.
auto	A logical indicating whether the Qualification is automatically granted to workers who request it. Default is FALSE.
auto.value	An optional parameter specifying the value that is automatically assigned to workers when they request it (if the Qualification is automatically granted).
verbose	Optionally print the results of the API request to the standard output. Default is taken from getOption('MTurkR.verbose', TRUE).
...	Additional arguments passed to request.

Details

A function to update the characteristics of a QualificationType. Name and keywords cannot be modified after a QualificationType is created.

updatequal() is an alias.

Value

A dataframe containing the QualificationTypeId of the newly created QualificationType and other details as specified in the request.

Author(s)

Thomas J. Leeper

References

API Reference

See Also

GetQualificationType

CreateQualificationType

DisposeQualificationType

SearchQualificationTypes

Examples

```
## Not run:
qual1 <-
CreateQualificationType(name="Worked for me before",
    description="This qualification is for people who have worked for me before",
    status = "Active",
    keywords="Worked for me before")
qual2 <-
UpdateQualificationType(qual1$QualificationTypeId,
    description="This qualification is for everybody!",
    auto=TRUE, auto.value="5")
DisposeQualificationType(qual1$QualificationTypeId)

## End(Not run)
```

Use Case: Categorization

Use Case: Categorization

Description

This page describes how to use MTurkR to collect categorization data

Details

This page describes how to use MTurkR to collect categorization data (perhaps for photo moderation or developing a training set) on Amazon Mechanical Turk. The workflow here is very similar to that for sentiment analyis (see sentiment).

The basic workflow for categorization is as follows:

1. Develope an HTML form template to display and record data
2. Optionally, develop a Qualification Test to select high-quality workers

3. Create a HITType (a display group) to organize the HITs you will produce
4. Create the HITs from the template and an input data.frame
5. Monitor and retrieve results

Creating a Categorization Template

MTurkR comes pre-installed with a HTMLQuestion HIT template showing the basic form of a categorization project (see `system.file("templates/categorization1.xml", package = "MTurkR")`. This template simply displays an image from a specified URL and asks workers to categorize the image as one of five things (person, animal, fruit, vegetable, or something else). Because an HTMLQuestion HIT is just an HTML document containing a form, we have one '`<input>`' field for each possible category with the name '`QuestionId1`' which is the data field we are interested in analyzing when we retrieve the results.

For your own project, you may want to add additional elements such as instructions to workers about how to perform the task, additional response options, or perhaps multiple dimensions on which to categorize (resolution, clarity, etc.). The key thing to remember is that the template must contain a template field which is what will be replaced by `BulkCreateFromTemplate` when you create the HITs. Note in the example template that the field is actually used in three places: (1) as the '`src`' of the HTML image field, (2) as the alt display text (in case the image doesn't load correctly), and (3) in a hidden form field. The last of these ensures that we are able to quickly and easily map the particular images into the categorization results returned by MTurk. If we don't do this, we will need to `merge` the results data with information about what image was displayed to each worker (because of an unfortunate feature of the MTurk API), so this saves us time later on.

Setting up the Project

Once the template is ready to go, we should think about who we want to complete the categorization task. By default, all HITs are available to all workers regardless of geography, language, experience, or quality. A standard practice is to limit HITs geographically and based on their past approval rating. We can specify this using `GenerateQualificationRequirement`:

```
q1 <- GenerateQualificationRequirement(c("Locale","Approved"),
                                       c("==",">"),
                                       c("US",90),
                                       preview = TRUE)
```

We can use this QualificationRequirement structure to limit our tasks to U.S.-based workers with greater than 90% past approval. The `preview = TRUE` argument means that ineligible workers will not even be able to preview the task.

An alternative approach is actually to create our own Qualification with a test that assesses workers' ability to correctly categorize, possibly while training them to do. A nice feature of such tests is that they can be automatically scored by the MTurk system so there is no need for you to actively manage the pool of eligible workers. This can be combined with the location and other QualificationRequirements just mentioned.

To create a Qualification test requires using some proprietary XML markup called "QuestionForm". This is basically an HTML-like form that specifies different kinds of questions and answer options, which will be displayed to workers interested in obtaining the qualification. An example form is included with MTurkR as `system.file("templates/qualificationtest1.xml", package = "MTurkR")`. I will leave it as an exercise to readers to understand the particularlities of the format.

To leverage the automatic scoring of a qualification test also requires creating an AnswerKey that maps answers in the Qualification test to an overall score. A possible AnswerKey for the Qualification test just mentioned is installed as: `system.file("templates/answerkey1.xml", package = "MTurkR")`.

To use these as QualificationRequirements for a task, we need to first create an MTurk QualificationType using the test and AnswerKey:

```
qf <- paste0(readLines(system.file("qualificationtest1.xml", package = "MTurkR")), collapse="")
qa <- paste0(readLines(system.file("answerkey1.xml", package = "MTurkR")), collapse="")
qual1 <- CreateQualificationType(name = "Qualification with Test",
        description = "This qualification is a demo",
        test = qf,
        answerkey = qa,
        status = "Active",
        keywords = "test, autogranted")
```

The QualificationType is now registered in the MTurk system and we can use it to create a QualificationRequirement to attach to our categorization HITs (e.g., to only include workers scoring above 50% on the test:

```
q2 <- GenerateQualificationRequirement(c("Locale","Approved",qual1$QualificationTypeId),
                                       c("==", ">", ">"),
                                       c("US", 90, 50),
                                       preview = TRUE)
```

Creating the HITs

To actually create the HITs, we need to define some "HITType" parameters that regulate what workers can complete the task, how much they will be paid, and what the task will look like when workers search for it on the MTurk worker interface.

```
newhittype <-
RegisterHITType(title = "Categorize an Image",
                description = "Categorize an image according to its content",
                reward = ".05",
                duration = seconds(hours=1),
                auto.approval.delay = seconds(days = 1),
                qual.req = q2,
                keywords = "categorization, image, coding, rating, sorting")
```

This creates a HITType, which is essentially a group of HITs; all HITs with the same HITType will be displayed together, allowing a worker to quickly complete many of the tasks in sequence. The value of `auto.approval.delay` is important; because we might be creating a very large number of HITs, this specifies that workers' answers will be automatically approved after a specific amount of time, preventing us from having to manually approve assignments.

With the HITType created, we can now create the HITs. The HIT template we are using specifies one template field, '`imageurl`', which is shown to the workers and also used to set the value of an HTML form field. This means that to use the template, we need to have a data.frame that contains one column, one row per image URL. MTurk does not return information about the HIT itself (e.g., what text was shown) with the results data, so taking this step of creating a hidden HTML form field recording that records the identifier thereby provides a simple way of mapping the displayed text to particular data we will retrieve from MTurk later on.

We can then use `BulkCreateFromTemplate` to process the data.frame and produce the HITs. (In

this example, the task contains very limited instructions, so in an applied case you may want to provide further details to workers about how to successfully complete the task.)

```
# template file
f <- system.file("templates/categorization1.xml", package = "MTurkR")
# input data.frame
dat <-
data.frame(imageurl = c("http://example.com/image1.png", "http://example.com/image1.png", "http://example.com/image1.p
      stringsAsFactors = FALSE)
# create the HITs (this is time consuming)
bulk <-
BulkCreateFromHITLayout(template = f,
                        input = dat,
                        hit.type = newhittype$HITTypeId,
                        assignments = 3,
                        annotation = paste("Image Categorization", Sys.Date()),
                        expiration = seconds(days = 7))
```

(Note: This does not actually upload images anywhere, so the value sof '`imageurl`' should be live URLs. If you have files stored locally, you will need to upload them to a high-capacity server first (perhaps Amazon S3).)

The code above processes the template using an input data.frame and creates a HIT for each row in the data.frame. In this case, we have asked for 3 assignments, meaning we would like three workers to categorize each image. The `bulk` object will be a data.frame containing one row per HIT. To perform operations on the individual HITs (e.g., expiring them early, retrieving results data, etc.) we will need the `HITId` value for each HIT as stored in this data.frame.

Monitoring Project

Once the HITs are live, they will likely be completed very quickly. If you have a very large project, however, it may take some time. To monitor the status of a project, you can use:

```
HITStatus(annotation = paste("Image Categorization", Sys.Date())
```

or

```
HITStatus(hit.type = newhittype$HITTypeId)
```

With a large number of HITs, the output of this will be fairly verbose, so you may instead simply want to examine the response objects themselves.

You may find a need to cancel HITs or change them in some way and there are numerous "maintenance" functions available to do this: `ExpireHIT`, `ExtendHIT` (to add assignments or time), `ChangeHITType` (to change display properties or payment of live HITs), and so forth. All of these functions can be called on a specific HIT, a HITType group, or on a set of HITs identified by their `annotation` value (which is a hidden field that can be useful for keeping track of HITs).

Retrieving Results

The final step of a categorization project is simply to retrieve the results and analyze them. Because we used a HITType and `BulkCreateFromTemplate`, retrieving the data for every assignment for every HIT in the project is very simple, but may be somewhat time consuming depending on the number of assignments involved. All we have to do is:

```
a <- GetAssignments(hit.type = newhittype$HITTypeId, return.all = TRUE)
```

If you have multiple projects under the same HITType (e.g., posted on different days), you can also retrieve assignments using the `annotation` field:
a <- GetAssignments(annotation = paste("Image Categorization", Sys.Date()), return.all = TRUE)

The response object is simply a data.frame. This data.frame will contain metadata about the task (HITTypeId, HITId, AssignmentId, WorkerId, submission and approval times) and values corresponding to every field in the HIT form. In this case, we will have two "answer" fields: '`QuestionId1`', which contains the score given by the workers, and '`imageurl`', providing the unique identifier for every image. Response are always stored as text because of the nature of the MTurk API, so you may need to coerce the data to a different type before performing some kind of analysis.

Author(s)

Thomas J. Leeper

See Also

For guidance on some of the functions used in this tutorial, see:

- `CreateQualificationType`
- `GenerateQualificationRequirement`
- `RegisterHITType`
- `BulkCreate`
- `HITStatus`
- `GetAssignments`

For some other tutorials on how to use MTurkR for specific use cases, see the following:

- survey, for collecting survey(-experimental) data
- sentiment, for doing sentiment coding
- webscraping, for manual scraping of web data

`Use Case: Scraping` *Use Case: Scraping*

Description

Use MTurkR to manually scrape web pages or documents

Details

This page describes how to use MTurkR to scrape human-readable data from the web or other sources using Amazon Mechanical Turk. While webscraping packages (such as XML, xml2, rvest, RSelenium) make it easy to retrieve structured web data and OCR technologies make it possible to extract text from PDFs and images, there are some cases in which data is stored in forms that are primarily human-readable rather than machine-readable. This particularly the case when information is textual but unstructured such as information stored in ad-hoc formats across different web pages (e.g., contact information for businesses) or in formats that are difficult or annoying to automatically scarpe (e.g., PDF). This tutorial describes how to use MTurkR to gather data from these kinds of sources.

Designing a HIT

The first step for a scraping project is to identify sources of information and decide what data you need to extract from those sources. Given the potentially massive variation in specific use cases, this tutorial will not address any particular solution in-depth. Once you have identified what information you would like to extract from the documents, you have to create a form representation in which the MTurk workers will submit information for a given document. This can be a QuestionForm (see `CreateHIT`), which is a proprietary markup, or more easily just an HTML form (see `GenerateHTMLQuestion`). An example of QuestionForm markup is given in `system.file("templates/tictactoe.xml", package = "MTurkR")` and an example of an HTMLQuestion markup is given in `system.file("templates/htmlquestion3.xml", package = "MTurkR")`. Either is perfectly acceptable. For those with HTML experience, HTMLQuestion is the easier approach; for those interested in making a very simple HIT and who have no previous HTML experience, QuestionForm may be easier to work with.

When designing the HIT, it should either contain a template placeholder (see an example in `system.file("templates/htmlq`
or you should create a separate HIT source file for every document or webpage that you wanted to scrape. Templating is simple if, for example, you want to scrape information from a list of websites because you can create a basic template and then use `BulkCreateFromTemplate` to add the website URL (and perhaps other details) to the HIT. If you are scraping a smaller number of documents that are quite irregular in format, it may be easier to manually create each HIT, save it as an .html file, and then create using either `BulkCreate` (if you store the .html files locally) or `BulkCreateFromURLs` if you upload those files to a remote server (e.g., Amazon S3).

The remainder of this tutorial will assume you have created the files manually and either wish to create HITs from the local files or from publicly accessible URLs for the server to which those HIT files have been uploaded. If neither of these cases applies (e.g., you want to use a HIT template), consider reading the "Use Case: Categorization" tutorial (see `categorization`).

Creating the HITs

If you have the HIT files saved locally, you will use `BulkCreate` to upload the HIT file to the MTurk server and use that file to create a single HIT. Begin by storing all of these files in a single folder. You can use `list.files` to retrieve all of the files and format them correctly for use in `BulkCreate`. From there, you will create a vector of character strings containing the files and pass tehse to `BulkCreate`:

```
qvec <- sapply(list.files(pattern = ".html"), function(x) {
      paste0(readLines(x, warn = FALSE), collapse = "\n")
    })
```

```
# create a HIT from each question file
hits <- BulkCreate(questions = qvec,
                   annotation = "My First Scraping Project",
                   title = "Retrieve information from a website",
                   description = "Find business information for the business named in the HIT.",
                   assignments = 1,
                   reward = ".25",
                   expiration = seconds(days = 4),
                   duration = seconds(minutes = 5),
                   keywords = "scraping, information search, finding, business, details, contact")
```

(Notes: (1) In this example, you are asking for one assignment per HIT. If each HIT is a business you want information for, this will allow one worker to retrieve information for each business. (2) This does not specify any constraints on which workers can complete HITs. If you want to restrict this to particular locations or by other measures, you will need to add a QualificationRequirement to the project (see `GenerateQualificationRequirement`).)

The above code will return a list, where each element is a one-row data.frame (or a representation of an API error message). Assuming all HITs were successfully created, you can then convert this to a single data.frame using `do.call("rbind", hits)`. You will want to preserve this data.frame as it contains the HITId value for each HIT, which you can use to manage the HITs.

Retrieving Results

Once all data have been scraped, you can retrieve the results using `GetAssignments`. (To know if the project is completed, you can monitor it using `HITStatus`.) `GetAssignments` can retrieve data for a single assignment, a single HIT, all HITs in a "HITType" group, or all HITs with a shared `annotation` value.

You may also find a need to cancel HITs or change them in some way and there are numerous "maintenance" functions available to do this: `ExpireHIT`, `ExtendHIT` (to add assignments or time), `ChangeHITType` (to change display properties or payment of live HITs), and so forth. All of these functions can be called on a specific HIT, a HITType group, or on a set of HITs identified by their `annotation` value (which is a hidden field that can be useful for keeping track of HITs).

In the above code, we set `annotation = "My First Scraping Project"` so we can use this to both monitor the project and retrieve assignment data:

```
# check status of project
HITStatus(annotation = "My First Scraping Project")
# get all assignment data
a <- GetAssignments(annotation = "My First Scraping Project", return.all = TRUE)
```

Here, `a` will be a data.frame containing information about each assignment. This data.frame will contain metadata about the task (HITTypeId, HITId, AssignmentId, WorkerId, submission and approval times) and values corresponding to every field in the HIT form. Response are always stored as text because of the nature of the MTurk API, so you may need to coerce the data to a different type before performing some kind of analysis.

Author(s)

Thomas J. Leeper

See Also

For guidance on some of the functions used in this tutorial, see:

- `BulkCreate`
- `HITStatus`
- `GetAssignments`

For some other tutorials on how to use MTurkR for specific use cases, see the following:

- survey, for collecting survey(-experimental) data
- categorization, for doing large-scale categorization (e.g., photo moderation or developing a training set)
- sentiment, for doing sentiment coding

`Use Case: Sentiment Analysis`

Use Case: Sentiment Analysis

Description

This page describes how to use MTurkR to collect sentiment data on Amazon Mechanical Turk.

Details

A common use case for MTurkR would be to code or score text or image data according to one or more dimensions. For example, you may have a large corpus of text that is divided into sentences and you want to code the sentiment of each sentence. Another would be that you want to score sentiment for messages posted on Twitter ("tweets"). This tutorial walks through how to create a simple sentiment analysis project and retrieve the results.

This tutorial is very similar to the one for image or text categorization (see categorization), though that tutorial also covers the use of Qualification Tests to selecting or training workers for a given task.

Creating a Sentiment Analysis HIT

MTurkR comes pre-installed with a HTMLQuestion HIT template for creating a sentiment project (see `system.file("templates/sentiment1.xml", package = "MTurkR")`. This template simply displays a message value and asks the MTurk worker to provide a sentiment score from -10 to +10 on a range indicator (or "slider"). Because an HTMLQuestion HIT is just an HTML document containing a form, we have one '`<input>`' field with the name '`sentiment`' which is the data field we are interested in analyzing when we retrieve the results.

The template works well with the `BulkCreateFromTemplate` function, which allows you to quickly produce a very large number of related HITs from a template and an input data.frame.

To begin, we need to define some HITType parameters that regulate what workers can complete the task, how much they will be paid, and what the task will look like when workers search for it on the MTurk worker interface.

```
newhittype <-
RegisterHITType(title = "Sentiment Analysis Task",
          description = "Score a tweet for positive or negative sentiment.",
          reward = ".05",
          duration = seconds(hours=1),
          auto.approval.delay = seconds(days = 1),
          keywords = "sentiment, coding, twitter, tweet, text, rating")
```

This creates a HITType, which is essentially a group of HITs; all HITs with the same HITType will be displayed together, allowing a worker to quickly complete many of the tasks in sequence. The value of `auto.approval.delay` is important; because we might be creating a very large number of HITs, this specifies that workers' answers will be automatically approved after a specific amount of time, preventing us from having to manually approve assignments.

Also, in this example, the task can be complete by any worker anywhere in the world; to change this, specify an object created by `GenerateQualificationRequirement` as the `qual.req` argument.

With the HITType created, we can now create the HITs. The HIT template we are using specifies one visible template field, '`tweettext`', which is shown to the workers and one hidden template field, '`tweetid`', which is hidden and that is also used to set the value of an HTML form field. This means that to use the template, we need to have a data.frame that contains two columns, one with each of those names. MTurk does not return information about the HIT itself (e.g., what text was shown) with the results data, so taking this step of creating a hidden HTML form field recording that records the identifier thereby provides a simple way of mapping the displayed text to particular sentiment data we will retrieve from MTurk later on.

We can then use `BulkCreateFromTemplate` to process the data.frame and produce the HITs. (In this example, the task contains very limited instructions, so in an applied case you may want to provide further details to workers about how to successfully complete the task.)

```
# template file
f <- system.file("templates/sentiment1.xml", package = "MTurkR")
# input data.frame
dat <-
data.frame(tweetid = 1:3,
       tweettext = c("abc", "def", "fgh"),
       stringsAsFactors = FALSE)
# create the HITs (this is time consuming)
bulk <-
BulkCreateFromHITLayout(template = f,
                input = dat,
                hit.type = newhittype$HITTypeId,
                assignments = 5,
                annotation = paste("Twitter Sentiment Project", Sys.Date()),
                expiration = seconds(days = 7))
```

Here we process the template using an input data.frame and create a HIT for each row in the data.frame. In this case, we have asked for 5 assignments, meaning we would like five workers to score each text. The `bulk` object will be a data.frame containing one row per HIT. To perform operations on the individual HITs (e.g., expiring them early, retrieving results data, etc.) we will need the `HITId` value for each HIT as stored in this data.frame.

Monitoring Project

Once the HITs are live, they will likely be completed very quickly. If you have a very large project, however, it may take some time. To monitor the status of a project, you can use:
HITStatus(annotation = paste("Twitter Sentiment Project", Sys.Date())

or
HITStatus(hit.type = newhittype$HITTypeId)

With a large number of HITs, the output of this will be fairly verbose, so you may instead simply want to examine the response objects themselves.

You may find a need to cancel HITs or change them in some way and there are numerous "maintenance" functions available to do this: `ExpireHIT`, `ExtendHIT` (to add assignments or time), `ChangeHITType` (to change display properties or payment of live HITs), and so forth. All of these functions can be called on a specific HIT, a HITType group, or on a set of HITs identified by their `annotation` value (which is a hidden field that can be useful for keeping track of HITs).

Retrieving Results

The final step of a sentiment analysis project is simply to retrieve the results and analyze them. Because we used a HITType and `BulkCreateFromTemplate`, retrieving the data for every assignment for every HIT in the project is very simple, but may be somewhat time consuming depending on the number of assignments involved. All we have to do is:
a <- GetAssignments(hit.type = newhittype$HITTypeId, return.all = TRUE)

If you have multiple projects under the same HITType (e.g., posted on different days), you can also retrieve assignments using the `annotation` field:
a <- GetAssignments(annotation = paste("Twitter Sentiment Project", Sys.Date()), return.all = TRUE)

The response object is simply a data.frame. This data.frame will contain metadata about the task (HITTypeId, HITId, AssignmentId, WorkerId, submission and approval times) and values corresponding to every field in the HIT form. In this case, we will have two "answer" fields: '`sentiment`', which contains the score given by the workers, and '`tweetid`', providing the unique identifier for every text. Response are always stored as text because of the nature of the MTurk API, so you may need to coerce the data to a different type before performing some kind of analysis.

Author(s)

Thomas J. Leeper

See Also

For guidance on some of the functions used in this tutorial, see:

- `BulkCreate`
- `RegisterHITType`
- `HITStatus`
- `GetAssignments`

For some other tutorials on how to use MTurkR for specific use cases, see the following:

- survey, for collecting survey(-experimental) data

- categorization, for doing large-scale categorization (e.g., photo moderation or developing a training set)
- webscraping, for manual scraping of web data

`Use Case: Surveys` *Use Case: Surveys*

Description

This page describes how to use MTurkR to collect survey(-experimental) data on Amazon Mechanical Turk.

Details

MTurk is widely used as a platform for social research, due to its low costs, ease of use, and quick access to participants. This use case tutorial walks through how to create MTurk HITs using MTurkR. It also describes some advanced features, such as managing panels of MTurk participants, and randomizing content.

Creating a survey HIT

Because a HIT is simply an HTML form, it is possible to design an entire survey(-experimental) questionnaire directly in MTurk. This could be done using an HTMLQuestion HIT. That would require, however, an extensive knowledge of HTML, likely a fair amount of javascript to display questions and validate worker responses, and a large amount of CSS styling. An alternative approach is to use an off-site survey tool (such as SurveyMonkey, Qualtrics, Google Spreadsheet Forms, etc.) to design a questionnaire and then simply use a HIT to direct workers to that task. There are two ways to do this in MTurkR. One is to create a "link and code" HIT that includes a link to an off-site tool and a text entry field for a worker to enter a completion code. The other is to embed the questionnaire directly as an ExternalQuestion HIT. The following two sections describe these approaches.

Link and code: To use the link and code, the HIT must contain the following:

- A link to the off-site survey
- A text entry form field into which the worker will enter a completion code
- Instructions to the worker about how to complete the HIT
- Optionally, javascript to dynamically display the link and/or entry field

MTurkR includes an XML file that could be used to create an HTMLQuestion "link and code" HIT. You can find it in `system.file("templates/htmlquestion1.xml", package = "MTurkR")`. A more robust example of an HTMLQuestion showing a survey link that is dynamically displayed once the HIT is accepted is included as `system.file("templates/surveylink.xml", package = "MTurkR")`. You can also create a HIT of this kind on the MTurk requester user interface (RUI) website.

ExternalQuestion: The link and code method is easier to setup, but invites problems with workers entering fake codes, workers completing the survey but not entering the code (and thus not getting paid), and other human error possibilities. By contrast, the ExternalQuestion method provides a seamless link between the MTurk interface and an off-site tool. Instead of showing the worker an HTML page with a link, configuring an ExternalQuestion actually shows the worker the survey directly inside an iframe in the MTurk worker interface. This eliminates the need for the requester to create codes and for workers to copy them.

The trade-off is that this method requires a somewhat sophisticated survey tool (e.g., Qualtrics) that can handle redirecting the worker back to the ExternalQuestion submit URL for their assignment. The submit URL has to take the form of: '`https://www.mturk.com/mturk/externalSubmit?assignmentId=work`
The '`foo=bar`' part is required by MTurk (it requires the assignmentId and at least one other field). Note: The previous link is for the live server. For testing in the requester sandbox, the base URL is: '`https://workersandbox.mturk.com/mturk/externalSubmit`'.

Creating the HIT itself is quite simple. It simply requires registering a HITType, building the ExternalQuestion structure (in which you can specify the height of the iframe in pixels), and using it in `CreateHIT`:

```
newhittype <-
RegisterHITType(title = "10 Question Survey",
          description = "Complete a 10-question survey about news coverage and your opinions",
          reward = ".20",
          duration = seconds(hours=1),
          keywords = "survey, questionnaire, politics")
eq <- GenerateExternalQuestion(url="https://www.example.com/myhit.html",frame.height="400")
CreateHIT(hit.type = newhittype$HITTypeId,
      question = eq$string,
      expiration = seconds(1),
      assignments = 1)
```

To make this work with a Qualtrics survey URL, do the following:

1. Create the Qualtrics survey
2. Setup the Qualtrics survey to extract embedded URL parameters (see documentation), which MTurk will attach to the Qualtrics survey link. You need to extract the '`assignmentId`' field. It is also helpful to extract the '`workerId`' field to embed it in the survey dataset as a unique identifier for each respondent.
3. Setup the Qualtrics survey to redirect (at completion) to the ExternalQuestion submit URL, using the '`assignmentId`' (the embedded data field) and at least one other survey field. See Qualtrics documentation for details. For the live server, this URL should look like: '`https://www.mturk.com/mturk/externalSubmit?assignmentId=${e://Field/assignmentId}&foo=bar`' and for the sandbox it should look like: '`https://workersandbox.mturk.com/mturk/externalSubmit?assignmen`
4. Create an ExternalQuestion HIT via MTurkR using the survey link from Qualtrics, as in the example above.

Managing a worker panel

The viability of MTurk as a platform for implementing social science experiments is well-understood, but the platform's comparative advantage for implementing complicated panel data collection is underappreciated, in part for technological reasons. The ability to recontact and pay large numbers of workers falls outside the functionality of the RUI. In this use case, the objective is to implement a

three-wave panel experiment, where respondents are randomly assigned to a condition at time 1, re-contacted to participate in a follow-up survey at time 2, and finally a second follow-up wave at time 3. The first stages of the workflow here are relatively easy. First, an experimental survey instrument is constructed with any tool (e.g., Qualtrics or one's own HTML). Second, a single HIT is created (without a qualification test) to which workers respond by completing the survey-experiment. Note that when only one HIT is needed, the parameters normally assigned with `RegisterHITType` can be specified in `CreateHIT`.

```
newhit <- CreateHIT(question = GenerateExternalQuestion("http://www.test.com/surveylink", 400)$string,
            title = "5-Question Survey",
            description = "Take a five-question survey about your political opinions.",
            reward = ".25",
            duration = seconds(hours=4),
            expiration = seconds(days=7),
            assignments = 1000,
            keywords = "survey, question, answers, research, politics, opinion",
            auto.approval.delay = seconds(days=2),
           qual.req = GenerateQualificationRequirement("Location","==","US",preview=TRUE))
```

Once a sufficient number of responses are collected (i.e., assignments completed), assignments can be reviewed such that only those who pass an attention check are approved and the remainder rejected:

```
review <- GetAssignments(hit=newhit$HITId)
correctcheck <- "7"
approve <- approve(assignments = review$AssignmentId[review$check==correct])
reject <- reject(assignments = review$AssignmentId[!review$check==correct])
```

After reviewing these assignments, MTurkR can be used to Bonus, Contact, and Qualify workers. To implement a panel, workers who completed the original HIT (the time 1 survey) are either granted a QualificationRequirement that gives them access to the second and third waves or they are simply contacted directly with a link to an off-site tool. If using the latter approach, the workers can be paid a bonus if they complete the follow-up surveys.

Using the first method, if we want to post the time 2 or time 3 surveys as new HITs but limit participation to those who have completed before, we would create a QualificationType using `CreateQualificationType`:

```
thenewqual <-
CreateQualificationType(
  name = "Completed Wave 1",
  description = "This qualification is for worker who completed wave 1.",
  status = 'Active',
  keywords = "Worked for me before",
  auto = FALSE)
```

Next, we can use `AssignQualification` to give all workers from your previous HIT a score above the threshold you'll use on the new HIT. So we'll set a score of 50 here, but you could use anything.

```
AssignQualification(
  qual = thenewqual$QualificationTypeId,
  workers = approve$WorkerId,
  value = 1)
```

This will return a data.frame specifying the scores assigned to each worker, so you don't need to store the result.

Finally, when we create the new HIT (for time 2 or time 3 surveys), we simply need to specify the QualificationType you created as a Qualification Requirement (using `GenerateQualificationRequirement`:

```
# generate a QualificationRequirement object
req <- GenerateQualificationRequirement(thenewqual$QualificationTypeId, "==", "1")
# create HIT
newhit <-
CreateHIT(
   title = "Survey",
   description = "5 question survey",
   reward = ".10",
   duration = seconds(days=1),
   keywords = "survey, questionnaire",
   expiration = seconds(days=7),
   assignments = 50,
   hitlayoutid = "23ZGOOGQSCM61T1H5H9U0U00OQWFFU",
   qual.req = req)
```

This restricts the survey to only be available to workers with the Qualification (i.e., those who completed the time 1 survey). Completions can then be monitored and approved in the same way as at time 1. The process could be repeated for the time 3 survey.

If we wanted to instead simply host the time 2 and time 3 surveys on an external site, we could invite workers from time 1 using `ContactWorkers` and then pay those who complete the survey using `GrantBonus`. We simply need to record which workers complete the survey. In the below example, we'll pass their WorkerId in the survey link, to record this automatically:

```
b1 <- "If you complete a follow-up survey, you will receive a $.50 bonus.\n
     You can complete the survey at the following link: http://www.test.com/link1?WorkerId="
t2 <- ContactWorkers(subjects = "Complete follow-up survey for $.50 bonus",
                     msgs = sapply(approve$WorkerId, FUN=function(worker) paste(b1, worker)),
                     workers = approve$WorkerId)
```

Once we have tracked completions, we can then pay bonuses by supplying a vector of workers and their corresponding '`AssignmentId`' (from the time 1 survey). This makes using `GrantBonus` a little clunky to use, but it works:

```
bonus <- GrantBonus(workers = review$WorkerId[review$WorkerId
                    assignments = review$AssignmentId[review$WorkerId
                    amounts = "1.00",
                    reasons = "Thanks for completing the follow-up survey!")
```

We can then repeat the process for the time 3 survey, using `ContactWorkers` and `GrantBonus` to invite and pay workers completing follow-up waves. The use of these tools, along with QualificationTypes, enables the maintenance of complex panels of workers with specific demographics or skills.

See Also

For guidance on some of the functions used in this tutorial, see:

- `CreateHIT`
- `GetAssignments`
- `ContactWorkers`

- GrantBonus
- CreateQualificationType

For some other tutorials on how to use MTurkR for specific use cases, see the following:

- categorization, for doing large-scale categorization (e.g., photo moderation or developing a training set)
- sentiment, for doing sentiment coding
- webscraping, for manual scraping of web data

`wizard.simple` *Interactive MTurkR Mode*

Description

An interactive, menu-based console wizard to perform MTurkR functions.

Usage

```
wizard.simple(graphics = FALSE, sandbox = NULL, ...)
```

Arguments

`graphics`	Optionally use graphical menus, if available, for the simple wizard. See `menu`. Default is `FALSE`.
`sandbox`	Optionally execute all requests in the MTurk sandbox rather than the live server. Default (in `MTurkR.Wizard`) is FALSE; the default in `wizard.simple` is NULL (with the wizard prompting for a value on load.
`...`	Additional arguments passed to `request`.

Details

An interactive, menu-based wizard (with optionally graphical menus) to perform most MTurkR operations. It is intended as a way for MTurk (and MTurkR) beginners to quickly create and monitor HITs; approve and reject assignments; notify, bonus, and block/unblock workers; manage Qualifications; monitor MTurk statistics; and interact with the MTurk Requester User Interface (RUI). All functionality accepts basic inputs interactively and executes requests without programming individual commands.

The wizard remains under development, but detailed documentation can be found at https://github.com/leeper/MTurkR/wiki/Wizard-Text-Based.

A more fully featured, interactive graphical wizard, which was previously available as part of MTurkR, has been refactored into a separate package called MTurkRGUI, available from CRAN. Please install and load that package for full documentation, or visit https://github.com/leeper/MTurkR/wiki/Wizard%20Graphical for more deatils.

Value

Currently returns nothing.

Author(s)

Thomas J. Leeper

Examples

```
## Not run:
wizard.simple()

## End(Not run)
```

XML *Parse MTurk XML to Dataframe*

Description

Parse MTurk XML Responses to R data.frames.

Usage

```
as.data.frame.AnswerKey(xml.parsed)
as.data.frame.Assignments(xml.parsed,
                          return.assignment.xml = FALSE)
as.data.frame.BonusPayments(xml.parsed)
as.data.frame.ExternalQuestion(xml.parsed)
as.data.frame.HITs(xml.parsed,
                   return.hit.xml = FALSE,
                   return.qual.list = TRUE,
                   sandbox = getOption('MTurkR.sandbox'))
as.data.frame.HTMLQuestion(xml.parsed)
as.data.frame.QualificationRequests(xml.parsed)
as.data.frame.QualificationRequirements(xml.parsed = NULL,
                                        xmlnodeset = NULL,
                                        hit.number = NULL,
                                        sandbox = getOption('MTurkR.sandbox'))
as.data.frame.Qualifications(xml.parsed)
as.data.frame.QualificationTypes(xml.parsed)
as.data.frame.QuestionFormAnswers(xml.parsed)
as.data.frame.QuestionForm(xml.parsed)
as.data.frame.ReviewResults(xml.parsed)
as.data.frame.WorkerBlock(xml.parsed)
```

Arguments

`xml.parsed`	A full MTurk XML response parsed by the `xmlParse`.
`xmlnodeset`	An XML nodeset.
`return.assignment.xml`	A logical indicating whether workers' responses to HIT questions should be returned.
`return.hit.xml`	A logical indicating whether the HIT XML should be returned. Default is `FALSE`.
`return.qual.list`	A logical indicating whether the QualificationRequirement list should be returned. Default is `TRUE`.
`hit`	An optional parameter included for advanced users, to return only one of the specified HITs.
`hit.number`	An optional parameter included for advanced users, to return only one of the specified HITs.
`sandbox`	A logical indicating whether GetQualificationType, called internally, should query the sandbox for user-defined QualificationTypes.

Details

Mostly internal functions to convert XML-formatted MTurk responses into more useful R dataframes. These are mostly internal to the extent that most users will never call them directly, but they may be useful if one needs to examine information stored in the MTurkR log file, or if `request` is used directly.

Value

A dataframe (or list of dataframes, in some cases) containing the request data.

Author(s)

Thomas J. Leeper

References

API Reference: Data Structures

Index