Package ‘googleAnalyticsR’

October 7, 2021

Type Package

Version 1.0.1

Title Google Analytics API into R

Description Interact with the Google Analytics APIs <https://developers.google.com/analytics/>, including the Core Reporting API (v3 and v4), Management API, User Activity API, GA4’s Data API and Admin API and Multi-Channel Funnel API.

URL https://code.markedmondson.me/googleAnalyticsR/

BugReports https://github.com/MarkEdmondson1234/googleAnalyticsR/issues

Depends R (>= 3.3.0)

Imports assertthat (>= 0.2.0), cli (>= 2.0.2), dplyr (>= 0.8.0), googleAuthR (>= 1.4.0), httr (>= 1.3.1), jsonlite (>= 1.5), magrittr (>= 1.5), measurementProtocol, memoise, methods, purrr (>= 0.2.2), rlang (>= 0.4.7), stats, tibble (>= 2.0.1), tidyr (>= 1.0.0), usethis, utils, whisker

Suggests bigQueryR (>= 0.3.1), covr, formatR, ganalytics, googleCloudStorageR (>= 0.2.0), htmlwidgets, knitr, lifecycle (>= 1.0.0), miniUI (>= 0.1.1), rmarkdown, shiny (>= 1.6.0), testthat

License MIT + file LICENSE

LazyData TRUE

RoxygenNote 7.1.1

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

NeedsCompilation no

Author Mark Edmondson [aut, cre] (<https://orcid.org/0000-0002-8434-3881>), Artem Klevtsov [ctb], Johann deBoer [ctb], David Watkins [ctb],
Olivia Brode-Roger [ctb],
Jas Sohi [ctb],
Zoran Selinger [ctb],
Octavian Corlade [ctb],
Maegan Whytock [ctb]

Maintainer  Mark Edmondson <m@sunholo.com>
Repository   CRAN
Date/Publication  2021-10-07 08:10:10 UTC

R topics documented:

accountPickerUI .................................................. 4
authDropdown ........................................................ 5
authDropdownUI ..................................................... 6
dim_filter ........................................................... 7
filter_clause_ga4 .................................................. 8
ga_accounts ......................................................... 10
nga_account_list .................................................. 11
ga_adwords .......................................................... 12
ga_adwords_add_linkid ........................................... 12
ga_adwords_delete_linkid ......................................... 13
ga_adwords_list ................................................... 14
ga_aggregate ........................................................ 15
ga_allowed_metric_dim .......................................... 16
ga_auth .............................................................. 16
ga_auth_setup ...................................................... 18
ga_cache_call ...................................................... 19
ga_clientid_activity .............................................. 19
ga_clientid_activity_unnest .................................... 21
ga_clientid_deletion .............................................. 22
ga_clientid_hash ................................................... 24
ga_custom_datasource ............................................. 24
ga_custom_upload .................................................. 25
ga_custom_upload_delete ......................................... 26
ga_custom_upload_file ............................................ 27
ga_custom_upload_list ............................................ 29
ga_custom_vars ..................................................... 30
ga_custom_vars_create .......................................... 30
ga_custom_vars_list .............................................. 32
ga_custom_vars_patch ............................................ 33
ga_data .............................................................. 34
ga_data_aggregations ............................................ 36
ga_data_filter ..................................................... 37
ga_data_order ..................................................... 40
ga_experiment ....................................................... 42
nga_experiment_list .............................................. 42
nga_filter .......................................................... 43
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ga_filter_add</td>
<td>43</td>
</tr>
<tr>
<td>ga_filter_apply_to_view</td>
<td>46</td>
</tr>
<tr>
<td>ga_filter_delete</td>
<td>46</td>
</tr>
<tr>
<td>ga_filter_list</td>
<td>47</td>
</tr>
<tr>
<td>ga_filter_update</td>
<td>48</td>
</tr>
<tr>
<td>ga_filter_update_filter_link</td>
<td>49</td>
</tr>
<tr>
<td>ga_filter_view</td>
<td>50</td>
</tr>
<tr>
<td>ga_filter_view_list</td>
<td>51</td>
</tr>
<tr>
<td>ga_goal</td>
<td>52</td>
</tr>
<tr>
<td>ga_goal_add</td>
<td>52</td>
</tr>
<tr>
<td>ga_goal_list</td>
<td>55</td>
</tr>
<tr>
<td>ga_goal_update</td>
<td>55</td>
</tr>
<tr>
<td>ga_meta</td>
<td>57</td>
</tr>
<tr>
<td>ga_model</td>
<td>58</td>
</tr>
<tr>
<td>ga_model_edit</td>
<td>59</td>
</tr>
<tr>
<td>ga_model_example</td>
<td>60</td>
</tr>
<tr>
<td>ga_model_load</td>
<td>61</td>
</tr>
<tr>
<td>ga_model_make</td>
<td>61</td>
</tr>
<tr>
<td>ga_model_refresh</td>
<td>63</td>
</tr>
<tr>
<td>ga_model_save</td>
<td>64</td>
</tr>
<tr>
<td>ga_model_shiny</td>
<td>65</td>
</tr>
<tr>
<td>ga_model_shiny_load</td>
<td>68</td>
</tr>
<tr>
<td>ga_model_shiny_template</td>
<td>69</td>
</tr>
<tr>
<td>ga_model_write</td>
<td>70</td>
</tr>
<tr>
<td>ga_mp_cid</td>
<td>70</td>
</tr>
<tr>
<td>ga_mp_event</td>
<td>71</td>
</tr>
<tr>
<td>ga_mp_event_item</td>
<td>71</td>
</tr>
<tr>
<td>ga_mp_send</td>
<td>73</td>
</tr>
<tr>
<td>ga_remarketing_build</td>
<td>75</td>
</tr>
<tr>
<td>ga_remarketing_create</td>
<td>77</td>
</tr>
<tr>
<td>ga_remarketing_estimate</td>
<td>78</td>
</tr>
<tr>
<td>ga_remarketing_get</td>
<td>79</td>
</tr>
<tr>
<td>ga_remarketing_list</td>
<td>79</td>
</tr>
<tr>
<td>ga_segment_list</td>
<td>80</td>
</tr>
<tr>
<td>ga_trackme</td>
<td>80</td>
</tr>
<tr>
<td>ga_unsampled</td>
<td>81</td>
</tr>
<tr>
<td>ga_unsampled_download</td>
<td>82</td>
</tr>
<tr>
<td>ga_unsampled_list</td>
<td>83</td>
</tr>
<tr>
<td>ga_users_add</td>
<td>84</td>
</tr>
<tr>
<td>ga_users_delete</td>
<td>85</td>
</tr>
<tr>
<td>ga_users_delete_linkid</td>
<td>86</td>
</tr>
<tr>
<td>ga_users_list</td>
<td>88</td>
</tr>
<tr>
<td>ga_users_update</td>
<td>89</td>
</tr>
<tr>
<td>ga_view</td>
<td>90</td>
</tr>
<tr>
<td>ga_view_list</td>
<td>91</td>
</tr>
<tr>
<td>ga_webproperty</td>
<td>92</td>
</tr>
<tr>
<td>ga_webproperty_list</td>
<td>92</td>
</tr>
<tr>
<td>googleAnalyticsR</td>
<td>93</td>
</tr>
</tbody>
</table>
accountPickerUI

Description

Makes a dropdown row for use for authentication with GA4 web properties.

Shiny Module for use with accountPickerUI

Usage

accountPickerUI(id, width = NULL, inColumns = FALSE)

accountPicker(id, ga_table, id_only = TRUE)

Arguments

- **id**: Shiny id
- **width**: The width of the input
- **inColumns**: Whether to wrap selectInputs in width=4 columns
- **ga_table**: A table GA4 accounts/web properties from `ga_account_summary("ga4")`
- **id_only**: Whether to return just the id, not the row
**authDropdown**

**Description**

Shiny Module for use with authDropdownUI

**Value**

If id_only=FALSE, the row of ga_table for the selected GA4 web property e.g. use ga_table$propertyId to send to ga_data calls. If id_only=TRUE, just the propertyId

**See Also**

Other Shiny modules: authDropdownUI(), authDropdown(), metricDimensionSelectUI(), multi_selectUI(), multi_select()

**Examples**

```r
## Not run:
ui <- fluidPage(title = "Shiny App",
    accountPickerUI("auth_menu", inColumns = TRUE))
server <- function(input, output, session){
  token <- gar_shiny_auth(session)
  accs <- reactive({
    req(token)
    ga_account_list("ga4")
  })
  property_id <- accountPicker("auth_menu", ga_table = accs, id_only = TRUE)
  shinyApp(gar_shiny_ui(ui, login_ui = silent_auth), server)
}
```
authDropdownUI

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>shiny input</td>
</tr>
<tr>
<td>output</td>
<td>shiny output</td>
</tr>
<tr>
<td>session</td>
<td>shiny session</td>
</tr>
<tr>
<td>ga.table</td>
<td>A table of GA tables</td>
</tr>
<tr>
<td>viewIdOnly</td>
<td>Default only returns the viewId, set to FALSE to return the row of ga.table satisfying the selections</td>
</tr>
<tr>
<td>rmNA</td>
<td>Will remove any rows that have NA listed for the columns. Set to FALSE to return all rows.</td>
</tr>
</tbody>
</table>

Details

Call via shiny::callModule(authDropdown,"your_id")

Value

GA View Id selected

See Also

Other Shiny modules: accountPickerUI(), authDropdownUI(), metricDimensionSelectUI(), multi_selectUI(), multi_select()

authDropdownUI authDropdown UI Shiny Module

Description

Makes a dropdown row for use for authentication.

Usage

authDropdownUI(id, width = NULL, inColumns = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Shiny id.</td>
</tr>
<tr>
<td>width</td>
<td>The width of the input</td>
</tr>
<tr>
<td>inColumns</td>
<td>whether to wrap selectInputs in width=4 columns.</td>
</tr>
</tbody>
</table>

Value

Shiny UI
dim_filter

Make a dimension filter object

Description

Make a dimension filter object

Usage

```r
dim_filter(
  dimension,
  operator = c("REGEXP", "BEGINS_WITH", "ENDS_WITH", "PARTIAL", "EXACT",
               "NUMERIC_EQUAL", "NUMERIC_GREATER_THAN", "NUMERIC_LESS_THAN", "IN_LIST"),
  expressions,
  caseSensitive = FALSE,
  not = FALSE
)
```

Arguments

- `dimension`: dimension name to filter on.
- `operator`: How to match the dimension.
- `expressions`: What to match. A character vector if operator is "IN_LIST"
- `caseSensitive`: Boolean.
- `not`: Logical NOT operator. Boolean.

Value

An object of class `dim_fil_ga4` for use in `filter_clause_ga4()`

See Also

Other filter functions: `filter_clause_ga4()`, `met_filter()`

Examples

```r
## Not run:
library(googleAnalyticsR)

## authenticate,
## or use the RStudio Addin "Google API Auth" with analytics scopes set
ga_auth()
```
## get your accounts
account_list <- google_analytics_account_list()

## pick a profile with data to query
ga_id <- account_list[23, 'viewId']

## create filters on metrics
mf <- met_filter("bounces", "GREATER_THAN", 0)
mf2 <- met_filter("sessions", "GREATER", 2)

## create filters on dimensions
df <- dim_filter("source", "BEGINS_WITH", "1", not = TRUE)
df2 <- dim_filter("source", "BEGINS_WITH", "a", not = TRUE)

## construct filter objects
fc2 <- filter_clause_ga4(list(df, df2), operator = "AND")
fc <- filter_clause_ga4(list(mf, mf2), operator = "AND")

## make v4 request
ga_data1 <- google_analytics_4(ga_id,
date_range = c("2015-07-30", "2015-10-01"),
dimensions = c("source", "medium"),
metrics = c("sessions", "bounces"),
met_filters = fc,
dim_filters = fc2,
filtersExpression = "ga:source!=\(direct\)"
)

## End(Not run)

---

**filter_clause_ga4**

*Make a dimension or metric filter clause object*

**Description**

Make a dimension or metric filter clause object

**Usage**

`filter_clause_ga4(filters, operator = c("OR", "AND"))`

**Arguments**

- **filters**
  - a list of `dim_filter` or `met_filter`. Only one type allowed.
- **operator**
  - combination of filter.
Details

If you have dimension and metric filters, make the clauses in two separate calls

Value

An object of class dim_fil_ga4 or met_fil_ga4

See Also

Other filter functions: dim_filter(), met_filter()

Examples

```r
## Not run:
library(googleAnalyticsR)

## authenticate,
## or use the RStudio Addin "Google API Auth" with analytics scopes set
ga_auth()

## get your accounts
account_list <- google_analytics_account_list()

## pick a profile with data to query
ga_id <- account_list[23,'viewId']

## create filters on metrics
mf <- met_filter("bounces", "GREATER_THAN", 0)
mf2 <- met_filter("sessions", "GREATER", 2)

## create filters on dimensions
df <- dim_filter("source","BEGINS_WITH","1",not = TRUE)
df2 <- dim_filter("source","BEGINS_WITH","a",not = TRUE)

## construct filter objects
fc2 <- filter_clause_ga4(list(df, df2), operator = "AND")
fic <- filter_clause_ga4(list(mf, mf2), operator = "AND")

## make v4 request
ga_data1 <- google_analytics(ga_id,
date_range = c("2015-07-30","2015-10-01"),
dimensions=c('source','medium'),
metrics = c('sessions','bounces'),
met_filters = fc,
dim_filters = fc2,
filtersExpression = "ga:source!=(direct)"
)

## End(Not run)
```
**ga_accounts**

*List account metadata*

**Description**

This gets a list of account meta data, that can be used in other management API functions.

**Usage**

```r
ga_accounts()
```

**Details**

This gets the meta data associated with the accounts you have access to with your user. If you want all information such as web properties and viewIds, use `ga_account_list` instead.

**Value**

A `data.frame` with accountid, name, an R datetime object (POSIXct) when the account was created and last updated, and the effective permissions your user has for those accounts.

**See Also**

Other account structure functions: `ga_account_list()`, `ga_view_list()`, `ga_view()`, `ga_webproperty_list()`, `ga_webproperty()`

**Examples**

```r
## Not run:
library(googleAnalyticsR)
ga_auth()
ga_accounts()

## End(Not run)
```
**ga_account_list**

Account summary for all accounts available to your user

---

**Description**

This is the recommended way to get all your account details for your user, including the web property and View IDs. The $viewId column contains the ID you need for the data fetching functions such as `google_analytics`.

**Usage**

```r
ga_account_list(type = c("universal", "ga4", "data"))
```

**Arguments**

- `type`: Whether to get account summary from universal analytics of GA4 (App_Web) properties

**Details**

Get a summary of all your accounts, web properties and views your authenticated user can see.

**Value**

- a dataframe of all account, webproperty and view data

**See Also**

- [https://developers.google.com/analytics/devguides/config/mgmt/v3/mgmtReference/management/accountSummaries/list](https://developers.google.com/analytics/devguides/config/mgmt/v3/mgmtReference/management/accountSummaries/list)
- Other account structure functions: `ga_accounts()`, `ga_view_list()`, `ga_view()`, `ga_webproperty_list()`, `ga_webproperty()`

**Examples**

```r
## Not run:
library(googleAnalyticsR)
ga_auth()
al <- ga_account_list()
al$viewId

## get account summary of GA4 properties
ga_account_list("ga4")

## End(Not run)
```
ga_adwords add linkid

---

ga_adwords

*Get AdWords Link meta data*

---

**Description**

Get AdWords Link meta data

**Usage**

`ga_adwords(accountId, webPropertyId, webPropertyAdWordsLinkId)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>webPropertyAdWordsLinkId</td>
<td>AdWords Link Id</td>
</tr>
</tbody>
</table>

**Value**

AdWords Meta data

**See Also**

Other Google Ad management functions: `ga_adwords_add_linkid()`, `ga_adwords_delete_linkid()`, `ga_adwords_list()`

---

**ga_adwords_add_linkid**  *Creates a Google Analytics webProperty-Google Ads link*

---

**Description**

Creates a link between and Adwords (Google ads) account and a Google Analytics property so that Adwords data can be accessed via Google Analytics and vice versa.

**Usage**

`ga_adwords_add_linkid(adwordsAccountId, linkName, accountId, webPropertyId)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adwordsAccountId</td>
<td>the customer id of the Adwords account visible within the Adwords account UI on the top right corner -or accessible via the Adwords API</td>
</tr>
<tr>
<td>linkName</td>
<td>a user defined way to call the link between the Adwords and Google Analytics accounts</td>
</tr>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
</tbody>
</table>
**ga_adwords_delete_linkid**

**Value**
confirmation message if successful

**See Also**

- Google documentation
- Other Google Ad management functions: `ga_adwords_delete_linkid()`, `ga_adwords_list()`, `ga_adwords()`

**Examples**

```r
## Not run:
library(googleAnalyticsR)
ga_auth()

ga_adwords_add_linkid("280-234-7592", "Google Ads Link", "65973592", "UA-65973592-1")
## End(Not run)
```

---

**ga_adwords_delete_linkid**

*Deletes a Google Analytics webProperty-Google Ads link*

**Description**

Removes a link between a Adwords (Google ads) account and a Google Analytics property

**Usage**

```r
ga_adwords_delete_linkid(accountId, webPropertyId, webPropertyAdWordsLinkId)
```

**Arguments**

- `accountId`: Account Id
- `webPropertyId`: Web Property Id
- `webPropertyAdWordsLinkId`: `webPropertyAdWordsLinkId`

**Value**

HTTP Status Code 204 with empty response body, if successful

**See Also**

- Google documentation
- Other Google Ad management functions: `ga_adwords_add_linkid()`, `ga_adwords_list()`, `ga_adwords()`
Examples

## Not run:

```r
library(googleAnalyticsR)
ga_auth()

# get the ID of the Adwords- Google Analytics link that you want to delete
# ID corresponding to the webPropertyAdWordsLinkId field
ga_adwords_list(65973592, "UA-65973592-1")

ga_adwords_delete_linkid(65973592, "UA-65973592-1", "ezW2dyaiQcGheWRAo69nCw")

# check its gone
ga_adwords_list(65973592, "UA-65973592-1")
```

## End(Not run)

---

**ga_adwords_list**  
List AdWords

**Description**

List AdWords

**Usage**

```r
ga_adwords_list(accountId, webPropertyId)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
</tbody>
</table>

**Value**

AdWords Links

**See Also**

Other Google Ad management functions:  
`ga_adwords_add_linkid()`, `ga_adwords_delete_linkid()`, `ga_adwords()`
**ga_aggregate**

Aggregate a Google Analytics dataframe over inputted columns

---

**Description**

A helper function to aggregate over dimensions

**Usage**

```r
ga_aggregate(
    ga_data,
    agg_names = NULL,
    mean_regex = "^avg|^percent|Rate$|^CPC$|^CTR$|^CPM$|^RPC$|^ROI$|^ROAS$|Per"
)
```

**Arguments**

- **ga_data**: A dataframe of data to aggregate
- **agg_names**: The columns to aggregate over
- **mean_regex**: The regex for column names to do mean() rather than sum()

**Details**

Will auto select metrics if they are numeric class columns. Will auto perform mean aggregation if metric names match `mean_regex` argument. If `agg_names` is NULL will aggregate over all.

**Examples**

```r
## Not run:
# use `/grave.Var` aggregateGAData` so you can on the fly create summary data
ga_data <- google_analytics(81416156,
    date_range = c("10daysAgo", "yesterday"),
    metrics = "sessions", dimensions = c("hour","date"))

# if we want totals per hour over the dates:
ga_aggregate(ga_data[,c("hour","sessions")], agg_names = "hour")

# it knows not to sum metrics that are rates:
ga_aggregate(ga_data[,c("hour","bounceRate")], agg_names = "hour")
```

## End(Not run)
**ga_allowed_metric_dim**  
*Create named list of allowed GA metrics/dimensions*

**Description**

Create named list of allowed GA metrics/dimensions

**Usage**

```r
ga_allowed_metric_dim(
  type = c("METRIC", "DIMENSION"),
  subType = c("all", "segment", "cohort"),
  callAPI = FALSE
)
```

**Arguments**

- `type`  
  Type of parameter to create
- `subType`  
  to restrict to only those in this type
- `callAPI`  
  This will update the meta table (Requires online authorization)
  This is useful to expand goalXCompletions to all the possibilities, as well as restricting to those that variables that work with your API call.
  Use internal meta table, but you have option to update to the latest version.

**Value**

A named list of parameters for use in API calls

---

**ga_auth**  
*Authenticate with Google Analytics OAuth2*

**Description**

A wrapper for gar_auth and gar_auth_service

**Usage**

```r
ga_auth(token = NULL, email = NULL, json_file = NULL)
```

**Arguments**

- `token`  
  An existing token or file location of a token to authenticate with
- `email`  
  An existing cached email to authenticate with or TRUE to authenticate with only email available. If not set then you will get an interactive prompt asking you to choose which email to authenticate with.
- `json_file`  
  Authentication service key you have downloaded from your Google Project - an alternative to OAuth2 email authentication
Details

Run this function first time to authenticate with Google in your browser.

After initial authentication, your authentication details will be kept globally for use later, tied to your email, and the next time you authenticate you will be given a prompt to choose which email to authenticate from. Set `email="your@email.com"` to skip the interactive prompt.

Value

Invisibly, the token that has been saved to the session

Multiple accounts

You can authenticate with a new email for each account. Supply a different email to use those details for your session.

Service accounts

If you use the service account JSON, you will need to add the service account email to your Google Analytics users to see data e.g. `xxxx@yyyyy.iam.gserviceaccount.com`

Auto-authentication

You can choose to auto-authenticate by creating a Google OAuth service account JSON file.

Specify an environment variable in R via a `.Renviron` file or using `Sys.setenv` which points to the file location of your chosen authentication file. See `Startup`

Once you have set the environment variable `GA_AUTH_FILE` to a valid file location, the function will look there for authentication details upon loading the library meaning you will not need to call `ga_auth()` yourself as you would normally.

An example `.Renviron` file is below:

```
GA_AUTH_FILE = "/Users/bob/auth/googleAnalyticsR.json"
```

`GA_AUTH_FILE` can be a service account JSON ending with file extension `.json`. Make sure to give the service account email access to your Google Analytics account as mentioned above.

Your own Google Project

Be default the Google Project used is shared by all users, so you may find it runs out of API calls. To mitigate that, create your own Google Project and turn on the Analytics APIs.

The best way to do this is to use `gar_set_client` by downloading your JSON client credentials and setting them to be found on package startup via the `GAR_CLIENT_JSON` environment argument. See `?googleAuthR::gar_set_client` function help pages for details.

Or you can then copy your Google Cloud Project's client ID and client secret, to place in options or environment arguments (whichever is easiest)

The environment args are below. Similar to auto-authentication, you can place your entries in an `.Renviron` file

```
GA_CLIENT_ID="XXXX" GA_CLIENT_SECRET="XXX" GA_WEB_CLIENT_ID="XXX" GA_WEB_CLIENT_SECRET="XXX" GA_USER_PROPERTY="XXX" GA_USER_METRIC="XXX"
```
Examples

## Not run:

# to use default package credentials (for testing)
library(googleAnalyticsR)
ga_auth()

# to use your own Google Cloud Project credentials
# go to GCP console and download client credentials JSON
# ideally set this in .Renviron file, not here but just for demonstration
Sys.setenv("GAR_CLIENT_JSON" = "location/of/file.json")
library(googleAnalyticsR)
# should now be able to log in via your own GCP project
ga_auth()

# reauthentication
# Once you have authenticated, set email to skip the interactive message
ga_auth(email = "my@email.com")

# or leave unset to bring up menu on which email to auth with
ga_auth()
# The googleAnalyticsR package is requesting access to your Google account.
# Select a pre-authorised account or enter '0' to obtain a new token.
# Press Esc/Ctrl + C to abort.
#1: my@email.com
#2: work@mybusiness.com
# you can set authentication for many emails, then switch between them e.g.
ga_auth(email = "my@email.com")
ga_account_list() # lists one set of accounts
ga_auth(email = "work@mybusiness.com")
ga_account_list() # lists second set of accounts

# or authenticate via the service key, that has been added to the GA as a user
nga_auth(json_file = "service-key.json")

## End(Not run)

---

**ga_auth_setup**  
Setup wizard for authentication options

### Description

Setup wizard for authentication options

### Usage

`ga_auth_setup()`
**ga_cache_call**

Setup caching of API calls

**Description**

Lets you cache API calls to disk

**Usage**

```r
ga_cache_call(cache_location)
```

**Arguments**

- `cache_location`: If RAM will save to memory, or specify a file folder location

**Details**

By default this is turned on upon package load to RAM. Should you want to cache calls to a folder then run this function to specify where.

---

**ga_clientid_activity**

User Activity Request

**Description**

Get activity on an individual user

**Usage**

```r
ga_clientid_activity(
    ids,
    viewId,
    id_type = c("CLIENT_ID", "USER_ID"),
    activity_type = NULL,
    date_range = NULL
)
```

**Arguments**

- `ids`: The userId or clientId. You can send in a vector of them
- `viewId`: The viewId
- `id_type`: Whether its userId or clientId
- `activity_type`: If specified, filters down response to the activity type. Choice between "PAGEVIEW", "SCREENVIEW", "GOAL", "ECOMMERCE", "EVENT"
- `date_range`: A vector of start and end dates. If not used will default to a week.
Details

The User Activity API lets you query an individual user’s movement through your website, by sending in the individual clientId or userId.

Bear in mind each call will count against your API quota, so fetching a large amount of client ids will be limited by that.

Use ga_clientid_activity_unnest to unnest deeply nested data in the hits data.

The timestamps are available to millisecond level but you will need to set your R options to see them e.g. options(digits.secs=3)

Value

A list of data.frames: $sessions contains session level data. $hits contains individual activity data

See Also

https://developers.google.com/analytics/devguides/reporting/core/v4/rest/v4/userActivity/search

Other clientid functions: ga_clientid_activity_unnest(), ga_clientid_deletion(), ga_clientid_hash()

Examples

```r
## Not run:

# access data for individual users
uar <- ga_clientid_activity(c("110690347.1461227730", "476443645.1541099566"),
    viewId = 81416156,
    date_range = c("2019-01-01","2019-02-01"))

# access clientIds for users who have transacted
viewId <- 106249469
date_range <- c("2019-01-01","2019-02-01")
cids <- google_analytics(viewId,
    date_range = date_range,
    metrics = "sessions",
    dimensions = "clientId",
    met_filters = filter_clause_ga4(
        list(met_filter("transactions",
          "GREATER_THAN",
          0)
    )))
transactors <- ga_clientid_activity(cids$clientId,
    viewId = viewId,
    date_range = date_range)

# access the data.frames returned:

# the session level data for the users passed in
uar$sessions
```
# the hit level activity for the users passed in
uar$hits

# filter the response to only include certain activity types, such as goals:
only_goals <- ga_clientid_activity(c("1106980347.1461227730", 
                                    "476443645.1541099566"),
                                    viewId = 81416156,
                                    date_range = c("2019-01-01", "2019-02-01"),
                                    activity_types = "GOAL")

## End(Not run)

--

ga_clientid_activity_unnest

Unnest user activity columns

**Description**

This helper function works with the output of user activity and parses out inner nested structure you may require.

Thanks to @jimmyg3g on GitHub for help with the ecommerce parsing.

**Usage**

```r
ga_clientid_activity_unnest(  
  hits,
  column = c("customDimension", "ecommerce", "goals")
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hits</td>
<td>The hits data.frame with the columns to expand</td>
</tr>
<tr>
<td>column</td>
<td>Which column to expand - one of &quot;customDimension&quot;,&quot;ecommerce&quot;,&quot;goals&quot;</td>
</tr>
</tbody>
</table>

**Details**

A function to help expand data out of nested columns returned by `ga_clientid_activity`

**Value**

An unnested data.frame tibble for all hits that matches the column

**See Also**

Other clientid functions: `ga_clientid_activity()`, `ga_clientid_deletion()`, `ga_clientid_hash()`
## Examples

```r
## Not run:
# access clientIds for users who have transacted
viewId <- 106249469
date_range <- c("2019-01-01","2019-02-01")
cids <- google_analytics(viewId,
    date_range = date_range,
    metrics = "sessions",
    dimensions = "clientId",
    met_filters = filter_clause_ga4(
        list(met_filter("transactions",
            "GREATER_THAN",
            0)
    ))
)
transactors <- ga_clientid_activity(cids$clientId,
    viewId = viewId,
    date_range = date_range)

# unnest ecommerce activity hits from users
ga_clientid_activity_unnest(transactors$hits, "ecommerce")

# unnest goal activity hits from users
ga_clientid_activity_unnest(transactors$hits, "goals")

# unnest custom dimension activity hits from users
ga_clientid_activity_unnest(transactors$hits, "customDimension")

## End(Not run)
```

---

ga_clientid_deletion  Create or update a user deletion request

### Description

The Google Analytics User Deletion API allows customers to process deletions of data associated with a given user identifier.

### Usage

```r
ga_clientid_deletion(
    userId,
    propertyId,
    idType = c("CLIENT_ID", "USER_ID", "APP_INSTANCE_ID"),
    propertyType = c("ga", "firebase")
)
```
Arguments

userId          A character vector of user ID's
propertyId      The Google Analytics Web property or Firebase ProjectId you are deleting the
                user from.
idType          Type of user. One of APP_INSTANCE_ID, CLIENT_ID or USER_ID.
propertyType    Firebase or Google Analytics

Details

The user explorer report in Google Analytics can give you the client.id you need to test.
A data deletion request can be applied to either a Google Analytics web property (specified by
propertyType="ga") or Firebase application (propertyType="firebase"). A user whose data
will be deleted can be specified by setting one of the identifiers the userId field. The type of the
identifier must be specified inside idType field.

There is a quota of 500 queries per day per cloud project.

The API returns a User Deletion Request Resource with deletionRequestTime field set. This field
is the point in time up to which all user data will be deleted. This means that all user data for the
specified user identifier and Google Analytics property or Firebase project will be deleted up to this
date and time - if the user with the same identifier returns after this date/time, they will reappear in
reporting.

Value

a data.frame with a row for each userID you sent in, plus a column with its deletionRequestTime

See Also

https://developers.google.com/analytics/devguides/config/userdeletion/v3/
Other clientid functions: ga_clientid_activity_unnest(), ga_clientid_activity(), ga_clientid_hash()

Examples

## Not run:

# make sure you are authenticated with user deletion scopes
options(googleAuthR.scopes.selected = "https://www.googleapis.com/auth/analytics.user.deletion")
ga_auth()

# a vector of ids
ids <- c("1489547420.1526330722", "1138076389.1526568883")

# do the deletions
ga_clientid_deletion(ids, "UA-1234-2")
# userId id_type property deletionRequestTime
ga_clientid_hash

Get hashed version of client id (also known as hashClientId, hashedClientId, or BigQuery's fullVisitorId)

Description
Get hashed version of client id (also known as hashClientId, hashedClientId, or BigQuery's fullVisitorId)

Usage
ga_clientid_hash(webPropertyId, clientId)

Arguments
- webPropertyId: Web Property Id
- clientId: Client Id

Value
hashedClientId object list

See Also
Other clientid functions: ga_clientid_activity_unnest(), ga_clientid_activity(), ga_clientid_deletion()
**ga_custom_upload**

**Details**

You primarily need this to get the customDataSourceId for the uploads via `ga_custom_upload_file`

**Value**

Custom Data Source

**See Also**

Other custom datasource functions: `ga_custom_upload_delete()`, `ga_custom_upload_file()`, `ga_custom_upload_list()`, `ga_custom_upload()`

---

**Description**

Get the status of a custom upload

**Usage**

```
    ga_custom_upload(
        accountId,  # Account Id
        webPropertyId,  # Web Property Id
        customDataSourceId,  # Custom data source Id
        uploadId,  # upload Id
        upload_object  # A custom upload Id object. Supply this or the other arguments.
    )
```

**Arguments**

- **accountId**: Account Id
- **webPropertyId**: Web Property Id
- **customDataSourceId**: Custom data source Id
- **uploadId**: upload Id
- **upload_object**: A custom upload Id object. Supply this or the other arguments.

**Details**

You can supply either upload_object generated via function or `ga_custom_upload_file`, or make an

**Value**

An object of class `ga_custom_data_source_upload`
See Also

Other custom datasource functions: \texttt{ga\_custom\_datasource()}. \texttt{ga\_custom\_upload\_delete()}. \texttt{ga\_custom\_upload\_file()}. \texttt{ga\_custom\_upload\_list()}

Examples

```r
## Not run:

upload_me <- data.frame(medium = "shinyapps",
                        source = "referral",
                        adCost = 1,
                        date = "20160801")

obj <- ga_custom_upload_file(47850439,
                             "UA-4748043-2",
                             ".jDsJHSFSU-uw038Bh8fUg",
                             upload_me)

## obj will initially have status = PENDING
obj

## Send obj to ga\_custom\_upload() to check and renew status
obj <- ga_custom_upload(upload_object = obj)
obj

## End(Not run)
```

\texttt{ga\_custom\_upload\_delete}

\textit{Deletes custom upload files for a given ids vector}

Description

Deletes custom upload files for a given ids vector
**Usage**

```r
ga_custom_upload_delete(
  accountId,
  webPropertyId,
  customDataSourceId,
  customDataImportUids
)
```

**Arguments**

- `accountId`  
  Account Id
- `webPropertyId`  
  Web Property Id
- `customDataSourceId`  
  Custom data source Id
- `customDataImportUids`  
  vector of file upload ids.

**See Also**


Other custom datasource functions:  
- `ga_custom_datasource()`, `ga_custom_upload_file()`, `ga_custom_upload_list()`, `ga_custom_upload()`

---

**ga_custom_upload_file**  
*Upload data to Google Analytics*

**Description**

Upload external data up to 1GB to Google Analytics via the management API.

**Usage**

```r
ga_custom_upload_file(accountId, webPropertyId, customDataSourceId, upload)
```

**Arguments**

- `accountId`  
  Account Id
- `webPropertyId`  
  Web Property Id
- `customDataSourceId`  
  Custom data source Id
- `upload`  
  An R data.frame or a file path location (character)
Details

You need to create a custom data source in the web UI first.

If you are uploading an R data frame, the function will prefix the column names with "ga: " for you if necessary.

After upload check the status by querying data sources using `ga_custom_upload` and examining the status field.

Currently only supports simple uploads (not resumable).

Value

An object of class `ga_custom_data_source_upload`

See Also

A guide for preparing the data is available: from Google here.
The dev guide for this function: Data Import Developer Guide
Other custom datasource functions: `ga_custom_datasource()`, `ga_custom_upload_delete()`, `ga_custom_upload_list()`, `ga_custom_upload()`

Examples

```r
## Not run:
upload_me <- data.frame(medium = "shinyapps",
  source = "referral",
  adCost = 1,
  date = "20160801")

obj <- ga_custom_upload_file(47850439,
  "UA-4748043-2",
  ".jDsJHSFSU-uw038Bh8fUG",
  upload_me)

## obj will initially have status = PENDING
obj

==Google Analytics Custom Data Source Upload==
Custom Data Source ID: _jDsJHSFSU-uw038Bh8fUG
Account ID: 47850439
Web Property Id: UA-4748043-2
Upload ID: 7yHLakeLSiK1zveVTiWZwA
Status: PENDING

## Send obj to ga_custom_upload() to check and renew status
obj <- ga_custom_upload(upload_object = obj)
obj

==Google Analytics Custom Data Source Upload==
Custom Data Source ID: _jDsJHSFSU-uw038Bh8fUG
```
Account ID: 47850439
Web Property Id: UA-4748043-2
Upload ID: 7yHLakeLSiK1zveVTiWZwA
Status: COMPLETED

## End(Not run)

---

### ga_custom_upload_list

**List Custom Data Source Uploads**

**Description**

List Custom Data Source Uploads

**Usage**

```python
ga_custom_upload_list(accountId, webPropertyId, customDataSourceId)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>customDataSourceId</td>
<td>Custom data source Id</td>
</tr>
</tbody>
</table>

**Value**

Custom Data Source Uploads List

**See Also**

Other custom datasource functions: `ga_custom_datasource()`, `ga_custom_upload_delete()`, `ga_custom_upload_file()`, `ga_custom_upload()`
ga\_custom\_vars \hspace{1cm} \textit{Get Custom Dimensions or Metrics}

\begin{description}
\item[Description]\hspace{1cm} Get Custom Dimensions or Metrics
\item[Usage]\hspace{1cm} \begin{verbatim}
ga\_custom\_vars(
    accountId,
    webPropertyId,
    type = c("customMetrics", "customDimensions"),
    customId
)
\end{verbatim}
\item[Arguments]\hspace{1cm} \begin{tabular}{ll}
    \textbf{accountId} & Account Id \\
    \textbf{webPropertyId} & Web Property Id \\
    \textbf{type} & A customMetric or customDimension \\
    \textbf{customId} & The customMetricId or customDimensionId \\
\end{tabular}
\item[Value]\hspace{1cm} Custom Metric or Dimension meta data
\item[See Also]\hspace{1cm} Other custom variable functions: \texttt{ga\_custom\_vars\_create()}, \texttt{ga\_custom\_vars\_list()}, \texttt{ga\_custom\_vars\_patch()}
\end{description}

\begin{description}
\item[Description]\hspace{1cm} Create a custom dimension by specifying its attributes.
\end{description}
**Usage**

```r
ga_custom_vars_create(
  name,
  index,
  accountId,
  webPropertyId,
  active,
  scope = c("HIT", "SESSION", "USER", "PRODUCT")
)
```

**Arguments**

- **name**: Name of custom dimension
- **index**: Index of custom dimension - integer between 1 and 20 (200 for GA360)
- **accountId**: AccountId of the custom dimension
- **webPropertyId**: WebPropertyId of the custom dimension
- **active**: TRUE or FALSE if custom dimension is active or not
- **scope**: Scope of custom dimension - one of "HIT", "SESSION", "USER", "PRODUCT"

**See Also**

- Custom dimensions support article
- Other custom variable functions: `ga_custom_vars_list()`, `ga_custom_vars_patch()`, `ga_custom_vars()`

**Examples**

```r
## Not run:
library(googleAnalyticsR)
ga_auth()

# create custom var
ga_custom_vars_create("my_custom_dim",
  index = 15,
  accountId = 54019251,
  webPropertyId = "UA-54019251-4",
  scope = "HIT",
  active = FALSE)

# view custom dimension in list
ga_custom_vars_list(54019251, webPropertyId = "UA-54019251-4", type = "customDimensions")
```
ga_custom_vars_list  List Custom Dimensions or Metrics

Description
List Custom Dimensions or Metrics

Usage

ga_custom_vars_list(
    accountId,  
    webPropertyId,  
    type = c("customDimensions", "customMetrics")
)

Arguments
accountId Account Id
webPropertyId Web Property Id
type A customMetric or customDimension

Details
This function lists all the existing custom dimensions or metrics for the web property.

Value
Custom Metric or Dimension List

See Also
Other custom variable functions: ga_custom_vars_create(), ga_custom_vars_patch(), ga_custom_vars()

Examples

## Not run:
library(googleAnalyticsR)
ga_auth()

ga_custom_vars_list(54019251, webPropertyId = "UA-54019251-4", type = "customDimensions")
ga_custom_vars_list(54019251, webPropertyId = "UA-54019251-4", type = "customMetrics")

## End(Not run)
ga_custom_vars_patch  Modify a custom dimension

Description

Modify existing custom dimensions

Usage

```
ga_custom_vars_patch(
  id,
  accountId,
  webPropertyId,
  name = NULL,
  active = NULL,
  scope = NULL,
  ignoreCustomDataSourceLinks = FALSE
)
```

Arguments

- **id**: The id of the custom dimension
- **accountId**: AccountId of the custom dimension
- **webPropertyId**: WebPropertyId of the custom dimension
- **name**: Name of custom dimension
- **active**: TRUE or FALSE if custom dimension is active or not
- **scope**: Scope of custom dimension - one of "HIT","SESSION","USER","PRODUCT"
- **ignoreCustomDataSourceLinks**: Force the update and ignore any warnings related to the custom dimension being linked to a custom data source / data set.

See Also

- Custom dimensions support article
- Other custom variable functions: `ga_custom_vars_create()`, `ga_custom_vars_list()`, `ga_custom_vars()`

Examples

```
# Not run:
library(googleAnalyticsR)
ga_auth()

# create custom var
ga_custom_vars_create("my_custom_dim",
  index = 7,
```
accountId = 54019251,
webPropertyId = "UA-54019251-4",
scope = "HIT",
active = FALSE)

# view custom dimension in list
ga_custom_vars_list(54019251, webPropertyId = "UA-54019251-4", type = "customDimensions")

# change a custom dimension
.ga_custom_vars_patch("ga:dimension7",
    accountId = 54019251,
    webPropertyId = "UA-54019251-4",
    name = "my_custom_dim2",
    active = TRUE)

# view custom dimensions again to see change
.ga_custom_vars_list(54019251, webPropertyId = "UA-54019251-4", type = "customDimensions")

## End(Not run)

---

**ga_data**

Google Analytics Data for GA4 (App+Web)

**Description**

**[Experimental]**

Fetches Google Analytics from the Data API for Google Analytics 4 (Previously App+Web)

**Usage**

```r
ga_data(
    propertyId,
    metrics,
    date_range = NULL,
    dimensions = NULL,
    dim_filters = NULL,
    dimensionDelimiter = "/",
    met_filters = NULL,
    orderBys = NULL,
    limit = 100,
    page_size = 100000L,
    realtime = FALSE,
    raw_json = NULL
)
```
Arguments

- **propertyId**: A GA4 property Id
- **metrics**: The metrics to request - see `ga_meta` - set to NULL to only see dimensions
- **date_range**: A vector with start and end dates in YYYY-MM-DD format - can send in up to four date ranges at once
- **dimensions**: The dimensions to request - see `ga_meta`
- **dim_filters**: Filter on the dimensions of the request - a filter object created by `ga_data_filter`
- **dimensionDelimiter**: If combining dimensions in one column, the delimiter for the value field
- **met_filters**: Filter on the metrics of the request - a filter object created by `ga_data_filter`
- **orderBys**: How to order the response - an order object created by `ga_data_order`
- **limit**: The number of rows to return - use -1 to return all rows
- **page_size**: The size of API pages - default is 100000L rows
- **realtime**: If TRUE then will call the real-time reports, that have a more limited set of dimensions/metrics - see valid real-time dimensions
- **raw_json**: You can send in the raw JSON string for a Data API request which will skip all checks

Details

This is the main function to call the Google Analytics 4 Data API.

Value

A data.frame tibble, including attributes metadata, metricAggregations and rowCount. Use `ga_data_aggregations` to extract the data.frames of metricAggregations

See Also

Documentation on Data API

Other GA4 functions: `ga_data_filter()`, `ga_data_order()`

Examples

```r
## Not run:

# send up to 4 date ranges
multi_date <- ga_data(
  206670707,
  metrics = c("activeUsers","sessions"),
  dimensions = c("date","city","dayOfWeek"),
  date_range = c("2020-03-31", "2020-04-27", "2020-04-30", "2020-05-27"),
  dim_filters = ga_data_filter("city"=="Copenhagen"),
  limit = 100
)
```
# metric and dimension expressions

# create your own named metrics
met_expression <- ga_data(
  206670707,
  metrics = c("activeUsers", "sessions", sessionsPerUser = "sessions/activeUsers"),
  dimensions = c("date", "city", "dayOfWeek"),
  date_range = c("2020-03-31", "2020-04-27"),
  limit = 100
)

# create your own aggregation dimensions
dim_expression <- ga_data(
  206670707,
  metrics = c("activeUsers", "sessions"),
  dimensions = c("date", "city", "dayOfWeek", cdow = "city/dayOfWeek"),
  date_range = c("2020-03-31", "2020-04-27"),
  limit = 100
)

# run a real-time report (no date dimension allowed)
realtime <- ga_data(
  206670707,
  metrics = "activeUsers",
  dimensions = c("city", "unifiedScreenName"),
  limit = 100,
  realtime = TRUE)

# extract meta data from the table
ga_data_aggregations(realtime)

# add ordering
a <- ga_data_order(-sessions)
b <- ga_data_order(-dayOfWeek, type = "NUMERIC")
ga_data(
  206670707,
  metrics = c("activeUsers", "sessions"),
  dimensions = c("date", "city", "dayOfWeek"),
  date_range = c("2020-03-31", "2020-04-27"),
  orderBys = c(a, b)
)

## End(Not run)
ga_data_filter

Description

[Experimental]
Metric aggregations are available in all requests. This function lets you easily access the data.frames

Usage

ga_data_aggregations(
  df,
  type = c("all", "totals", "maximums", "minimums", "count")
)

Arguments

df
  A data.frame result from ga_data

type
  totals, maximums, minimums, counts (if available) or all

Examples

## Not run:
##
### send up to 4 date ranges
multi_date <- ga_data(
  206670707,
  metrics = c("activeUsers","sessions"),
  dimensions = c("date","city","dayOfWeek"),
  date_range = c("2020-03-31", "2020-04-27", "2020-04-30", "2020-05-27"),
  dim_filters = ga_data_filter("city=="Copenhagen"),
  limit = 100
)

# metric aggregations for each date range
ga_data_aggregations(multi_date, type = "all")

# specify type
ga_data_aggregations(multi_date, type = "maximums")

## End(Not run)

---

**ga_data_filter**

Filter DSL for GA4 filters

Description

Use with ga_data to create filters

Usage

ga_data_filter(x)
Arguments

- x: Filter DSL enabled syntax or the output of a previous call to this function - see examples

Details

This uses a specific filter DSL syntax to create GA4 filters that can be passed to `ga_data` arguments `dim_filters` or `met_filters`. Ensure that the fields you use are either all metrics or all dimensions.

The syntax uses operators and the class of the value you are setting (string, numeric or logical) to construct the filter expression object.

Fields including custom fields for your propertyId can be imported if you fetch them via `ga_meta("data", propertyId = 12345)` before you construct a filter. If you do not want filters to be validated, then send them in as strings ("field").

The DSL rules are:

- Single filters can be used without wrapping in filter expressions
- A single filter syntax is `field` `(operator)` `value`  
- `field` is a dimension or metric for your web property, which you can review via `ga_meta`
- `field` can be validated if you fetch metadata before you construct the filter. If you do this, you can call the fields without quote strings e.g. `city` and not "city"
- `operator` for metrics can be one of: `==`, `>`, `>=`, `<`, `<=`
- `operator` for dimensions can be one of: `==`, `\%begins\%`, `\%ends\%`, `\%contains\%`, `\%in\%`, `\%regex\%`, `\%regex_partial\%`
- dimension `operator` are by default case sensitive. Make them case insensitive by using UPPER case variations `\%BEGINS\%`, `\%ENDS\%`, ... or `===` for exact matches
- `value` can be strings ("dim1"), numerics (55), string vectors (`c("dim1","dim2")`), numeric vectors (`c(1,2,3)`) or boolean (TRUE) - the type will create different types of filters - see examples
- Create filter expressions for multiple filters when using the operators: `&`, `|`, `!` for logical combinations of AND, OR and NOT respectively.

Value

A `FilterExpression` object suitable for use in `ga_data`

See Also

Other GA4 functions: `ga_data_order()`, `ga_data()`

Examples

```r
## Not run:
# start by calling `ga_meta("data")` to put valid field names in your environment
meta <- ga_meta("data")
```
# if you have custom fields, supply your propertyId to ga_meta()
custom_meta <- ga_meta("data", propertyId = 206670707)
custom_meta[grepl("^customEvent", custom_meta$apiName),]

## End(Not run)
## filter clauses

# OR string filter
ga_data_filter(city=="Copenhagen" | city == "London")
# inlist string filter
ga_data_filter(city==c("Copenhagen","London"))
# AND string filters
ga_data_filter(city=="Copenhagen" & dayOfWeek == "5")
# ! - invert string filter
ga_data_filter(!(city=="Copenhagen" | city == "London"))

# multiple filter clauses
f1 <- ga_data_filter(city==c("Copenhagen","London","Paris","New York") &
                     (dayOfWeek=="5" | dayOfWeek=="6"))

# build up complicated filters
f2 <- ga_data_filter(f1 | sessionSource=="google")
f3 <- ga_data_filter(f2 & !sessionMedium=="cpc")
f3

## numeric filter types

# numeric equal filter
ga_data_filter(sessions==5)
# between numeric filter
ga_data_filter(sessions==c(5,6))
# greater than numeric
ga_data_filter(sessions > 0)
# greater than or equal
nga_data_filter(sessions >= 1)
# less than numeric
nga_data_filter(sessions < 100)
# less than or equal numeric
nga_data_filter(sessions <= 100)

## string filter types

# begins with string
nga_data_filter(city %begins% "Cope")
# ends with string
nga_data_filter(city %ends% "hagen")
# contains string
nga_data_filter(city %contains% "ope")
# regex (full) string
nga_data_filter(city %regex% ^"Cope")
# regex (partial) string
nga_data_filter(city %regex_partial% "ope")

# by default string filters are case sensitive.
# Use UPPERCASE operator to make then case insensitive
# begins with string (case insensitive)
ga_data_filter(city %BEGINS% "cope")
# ends with string (case insensitive)
ga_data_filter(city %ENDS% "Hagen")
# case insensitive exact
 ga_data_filter(city %==% "coPENGhagen")

# avoid validation by making fields strings
 ga_data_filter("city" %==% "coPENGhagen")

gp_data_order

**Order DSL for GA4 OrderBy**

**Description**

Use with `ga_data` to create orderBys

**Usage**

```r
ga_data_order(
  x,
  type = c("ALPHANUMERIC", "CASE_INSENSITIVE_ALPHANUMERIC", "NUMERIC")
)
```

**Arguments**

- `x` Order DSL enabled syntax
- `type` Order Type

**Details**

The DSL rules are:

- Fields can be quoted or unquoted. If unquoted they will be validated
- Use `+` as a prefix to indicate ascending order e.g. `+sessions`
- Use `-` as a prefix to indicate decreasing order e.g. `-sessions`
- Combine order fields without commas e.g. `+sessions -city`
- Ordering of dimensions can also specify a type of ordering: ALPHANUMERIC, CASE_INSENSITIVE_ALPHANUMERIC, NUMERIC

The dimension ordering have these effects:

- ALPHANUMERIC For example, "2" < "A" < "X" < "b" < "z"
- CASE_INSENSITIVE_ALPHANUMERIC Case insensitive alphanumeric sort by lower case Unicode code point. For example, "2" < "A" < "b" < "X" < "z"
- NUMERIC Dimension values are converted to numbers before sorting. For example in NUMERIC sort, "25" < "100", and in ALPHANUMERIC sort, "100" < "25". Non-numeric dimension values all have equal ordering value below all numeric values
Value

A list of `OrderBy` objects suitable for use in `ga_data`

See Also

https://developers.google.com/analytics/devguides/reporting/data/v1/rest/v1alpha/OrderBy

Other GA4 functions: `ga_data_filter()`, `ga_data()`

Examples

```r
# session in descending order
ga_data_order(-sessions)

# city dimension in ascending alphanumeric order
ga_data_order(+city)

# as above plus sessions in descending order
ga_data_order(+city -sessions)

# as above plus activeUsers in ascending order
ga_data_order(+city -sessions +activeUsers)

# dayOfWeek dimension in ascending numeric order
ga_data_order(+dayOfWeek, type = "NUMERIC")

# you can also combine them with c()
a <- ga_data_order(-sessions)
b <- ga_data_order(-dayOfWeek, type = "NUMERIC")
c(a, b)

## Not run:
# example of use
ga_data(
  206670707,
  metrics = c("activeUsers","sessions"),
  dimensions = c("date","city","dayOfWeek"),
  date_range = c("2020-03-31", "2020-04-27"),
  orderBys = ga_data_order(-sessions -dayOfWeek)
)

## End(Not run)
```
**ga_experiment_list**

---

**ga_experiment**

*Experiments Meta data*

---

**Description**

Experiments Meta data

**Usage**

`ga_experiment(accountId, webPropertyId, profileId, experimentId)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>profileId</td>
<td>Profile Id</td>
</tr>
<tr>
<td>experimentId</td>
<td>Experiment Id</td>
</tr>
</tbody>
</table>

**Value**

Experiment Meta Data

**See Also**

Other managementAPI functions: `ga_experiment_list()`, `ga_filter_add()`, `ga_filter_apply_to_view()`, `ga_filter_update_filter_link()`, `ga_filter_update()`, `ga_segment_list()`

---

**ga_experiment_list**

*List Experiments*

---

**Description**

List Experiments

**Usage**

`ga_experiment_list(accountId, webPropertyId, profileId)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>profileId</td>
<td>Profile Id</td>
</tr>
</tbody>
</table>
**ga_filter**

**Value**
Experiments List

**See Also**
Other management API functions: `ga_experiment()`, `ga_filter_add()`, `ga_filter_apply_to_view()`, `ga_filter_update_filter_link()`, `ga_filter_update()`,
`ga_segment_list()`

---

**ga_filter**
Get specific filter for account

**Description**
Get specific filter for account

**Usage**
`ga_filter(accountId, filterId)`

**Arguments**
- `accountId`  Account Id
- `filterId`  Filter Id

**Value**
filter list

**See Also**
Other filter management functions: `ga_filter_delete()`, `ga_filter_list()`, `ga_filter_view_list()`, `ga_filter_view()`

---

**ga_filter_add**
Create a new filter and add it to the view (optional).

**Description**
Take a filter object and add and/or apply it so its live.
Usage

```r
ga_filter_add(
  Filter,
  accountId,
  webPropertyId = NULL,
  viewId = NULL,
  linkFilter = FALSE
)
```

Arguments

- **Filter**: The Filter object to be added to the account or view. See examples.
- **accountId**: Account Id of the account to add the Filter to
- **webPropertyId**: Property Id of the property to add the Filter to
- **viewId**: View Id of the view to add the Filter to
- **linkFilter**: If TRUE will apply the Filter to the view. Needs propertyId and viewId to be set.

Details

If you don’t set `linkFilter=TRUE` then the filter will only be created but not applied. You will find it listed in the admin panel Account > All Filters. You can then use `ga_filter_apply_to_view` to apply later on.

Value

The filterId created if `linkFilter=FALSE` or a Filter object if `linkFilter=TRUE`

See Also

- [https://developers.google.com/analytics/devguides/config/mgmt/v3/mgmtReference/#Filters](https://developers.google.com/analytics/devguides/config/mgmt/v3/mgmtReference/#Filters)
- Other managementAPI functions: `ga_experiment_list()`, `ga_experiment()`, `ga_filter_apply_to_view()`, `ga_filter_update_filter_link()`, `ga_filter_update()`, `ga_segment_list()`

Examples

```r
## Not run:
## Create a filter object for adding an IP exclusion:
Filter <- list(
  name = 'Exclude Internal Traffic',
  type = 'EXCLUDE',
  excludeDetails = list(
    field = 'GEO_IP_ADDRESS',
    matchType = 'EQUAL',
    expressionValue = '199.04.123.1',
    caseSensitive = 'False'
  )
)
```
# create and add the filter to the view specified
my_filter <- ga_filter_add(Filter,
    accountId = 12345,
    webPropertyId = "UA-12345-1",
    viewId = 654321,
    linkFilter = TRUE)

# only create the filter, don't apply it to any view - returns filterId for use later
my_filter <- ga_filter_add(Filter,
    accountId = 12345,
    linkFilter = FALSE)

## Other examples of filters you can create below:
## Create a filter object for making campaign medium lowercase
Filter <- list(
    name = 'Lowercase Campaign Medium',
    type = 'LOWERCASE',
    lowercaseDetails = list(
        field = 'CAMPAIGN_MEDIUM'
    )
)

## Create a filter object to append hostname to URI
Filter <- list(
    name = 'Append hostname to URI',
    type = 'ADVANCED',
    advancedDetails = list(
        fieldA = 'PAGE_HOSTNAME',
        extractA = '(.*',
        fieldARequired = 'True',
        fieldB = 'PAGE_REQUEST_URI',
        extractB = '(.*',
        fieldBRequired = 'False',
        outputConstructor = '$A1$B1',
        outputToField = 'PAGE_REQUEST_URI',
        caseSensitive = 'False',
        overrideOutputField = 'True'
    )
)

## Create a filter object to add www hostname without it
Filter <- list(
    name = 'Search and Replace www',
    type = 'SEARCH_AND_REPLACE',
    searchAndReplaceDetails = list(
        field = 'PAGE_HOSTNAME',
        searchString = '^exampleUSA\.com$',
        replaceString = 'www.exampleUSA.com',
        caseSensitive = 'False'
    )
)
ga_filter_apply_to_view

Apply an existing filter to view.

Description

Apply an existing filter to view.

Usage

\[ \text{ga_filter_apply_to_view(filterId, accountId, webPropertyId, viewId)} \]

Arguments

- filterId: The id of the filter to be added to profile/view
- accountId: Account Id of the account that contains the filter
- webPropertyId: Web property Id to create profile filter link for
- viewId: Profile/view Id to create profile filter link for

Value

A profileFilterLink object

See Also

Other managementAPI functions: \( \text{ga_experiment_list(), ga_experiment(), ga_filter_add(), ga_filter_update_filter_link(), ga_filter_update(), ga_segment_list()} \)

ga_filter_delete

Delete a filter from account or remove from view.

Description

Delete a filter from account or remove from view.

Usage

\[ \text{ga_filter_delete( accountId, webPropertyId = NULL, viewId = NULL, filterId, removeFromView = FALSE )} \]
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id of the account that contains the filter</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Property Id of the property that contains the filter</td>
</tr>
<tr>
<td>viewId</td>
<td>View Id of the view that contains the filter</td>
</tr>
<tr>
<td>filterId</td>
<td>Filter Id of the filter to be deleted</td>
</tr>
<tr>
<td>removeFromView</td>
<td>Default if FALSE. If TRUE, deletes the filter from the view</td>
</tr>
</tbody>
</table>

Value

TRUE if successful

See Also

Other filter management functions: ga_filter_list(), ga_filter_view_list(), ga_filter_view(), ga_filter()
Updates an existing filter.

Usage

ga_filter_update(Filter, accountId, filterId, method = c("PUT", "PATCH"))

Arguments

Filter The Filter object to be updated See examples from ga_filter_add()
accountId Account Id of the account that contains the filter
filterId The id of the filter to be modified
method PUT by default. For patch semantics use PATCH

Value

A filterManagement object

See Also

https://developers.google.com/analytics/devguides/config/mgmt/v3/mgmtReference/#Filters

Other managementAPI functions: ga_experiment_list(), ga_experiment(), ga_filter_add(),
ga_filter_apply_to_view(), ga_filter_update_filter_link(), ga_segment_list()

Examples

```r
## Not run:

# create a filter object
Filter <- list(
  name = 'googleAnalyticsR test1: Exclude Internal Traffic',
  type = 'EXCLUDE',
  excludeDetails = list(
    field = 'GEO_IP_ADDRESS',
    matchType = 'EQUAL',
    expressionValue = '199.04.123.1',
    caseSensitive = 'False'
  )
)

# add a filter (but don't link to a View)
filterId <- ga_filter_add(Filter,
  accountId = 123456,
  linkFilter = FALSE)
```
# change the name of the filter
change_name <- "googleAnalyticsR test2: Changed name via PATCH"

# using PATCH semantics, only need to construct what you want to change
filter_to_update <- list(name = test_name)

# update the filter using the filterId
ga_filter_update(filter_to_update, accountId2, filterId, method = "PATCH")

## End(Not run)

---

**ga_filter_update_filter_link**

Update an existing profile filter link. Patch semantics supported

**Description**

Update an existing profile filter link. Patch semantics supported

**Usage**

```r
ga_filter_update_filter_link(
  viewFilterLink, accountId, webPropertyId, viewId, linkId,
  method = c("PUT", "PATCH")
)
```

**Arguments**

- `viewFilterLink` The profileFilterLink object
- `accountId` Account Id of the account that contains the filter
- `webPropertyId` Web property Id to which the profile filter link belongs
- `viewId` View Id to which the profile filter link belongs
- `linkId` The id of the profile filter link to be updated
- `method` PUT by default. Supports patch semantics when set to PATCH

**See Also**

- Other managementAPI functions: `ga_experiment_list()`, `ga_experiment()`, `ga_filter_add()`, `ga_filter_apply_to_view()`, `ga_filter_update()`, `ga_segment_list()`
Examples

```r
## Not run:

# create a filter object
Filter <- list(
  name = 'googleAnalyticsR test: Exclude Internal Traffic',
  type = 'EXCLUDE',
  excludeDetails = list(
    field = 'GEO_IP_ADDRESS',
    matchType = 'EQUAL',
    expressionValue = '199.04.123.1',
    caseSensitive = 'False'
  )
)

# link Filter to a View
response <- ga_filter_add(Filter,
  accountId = 12345,
  webPropertyId = "UA-12345-1",
  viewId = 654321,
  linkFilter = TRUE)

# create Filter patch to move existing filter up to rank 1
viewFilterLink <- list(rank = 1)

# use the linkId given in response$id to update to new rank 1
response2 <- ga_filter_update_filter_link(viewFilterLink,
  accountId = 12345,
  webPropertyId = "UA-12345-1",
  viewId = 654321,
  linkId = response$id)

## End(Not run)
```

---

**ga_filter_view**  
*Get specific filter for view (profile)*

**Description**

Get specific filter for view (profile)

**Usage**

```r
ga_filter_view(accountId, webPropertyId, viewId, linkId)
```
**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>viewId</td>
<td>Profile Id</td>
</tr>
<tr>
<td>linkId</td>
<td>Link Id</td>
</tr>
</tbody>
</table>

**Value**

filter list

**See Also**

Other filter management functions: `ga_filter_delete()`, `ga_filter_list()`, `ga_filter_view_list()`, `ga_filter()`

---

**Description**

List filters for view (profile)

**Usage**

`ga_filter_view_list(accountId, webPropertyId, viewId)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>viewId</td>
<td>Profile Id</td>
</tr>
</tbody>
</table>

**Value**

filter list

**See Also**

Other filter management functions: `ga_filter_delete()`, `ga_filter_list()`, `ga_filter_view()`, `ga_filter()`
ga_goal

Get goal

Description
Get goal

Usage

```
ga_goal(accountId, webPropertyId, profileId, goalId)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>profileId</td>
<td>Profile Id</td>
</tr>
<tr>
<td>goalId</td>
<td>Goal Id</td>
</tr>
</tbody>
</table>

Value
Goal meta data

See Also
Other goal management functions: `ga_goal_add()`, `ga_goal_list()`, `ga_goal_update()`

ga_goal_add
Create a new goal.

description
Create a new goal.

Usage

```
ga_goal_add(Goal, accountId, webPropertyId, viewId)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>The Goal object to be added to the view. See examples.</td>
</tr>
<tr>
<td>accountId</td>
<td>Account Id of the account to add the Goal to</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Property Id of the property to add the Goal to</td>
</tr>
<tr>
<td>viewId</td>
<td>View Id of the view to add the Goal to</td>
</tr>
</tbody>
</table>
Value

The Goal object

See Also

https://developers.google.com/analytics/devguides/config/mgmt/v3/mgmtReference/#Goals

Other goal management functions: `ga_goal_list()`, `ga_goal_update()`, `ga_goal()`

Examples

```r
## Not run:

## Create a Goal object based on destination:
Goal <- list(
id = '17',
active = TRUE,
name = 'Checkout',
type = 'URL_DESTINATION',
urlDestinationDetails = list(
  url = 'checkout\thank_you',
  matchType = 'REGEX',
caseSensitive = FALSE,
firstStepRequired = FALSE,
steps = list(
  list(
    number = 1,
    name = 'Product',
    url = 'products\'
  ),
  list(
    number = 2,
    name = 'Cart',
    url = 'cart'
  ),
  list(
    number = 3,
    name = 'Contact',
    url = 'checkout\contact_information'
  ),
  list(
    number = 4,
    name = 'Shipping',
    url = 'checkout\shipping'
  ),
  list(
    number = 5,
    name = 'Payment',
    url = 'checkout\payment'
  ),
  list(
    number = 6,
```

```r
```
name = 'Processing',
url = 'checkout\processing'
)
)
)
)

## Create a Goal object based on an event:
Goal <- list(
  id = '9',
  active = TRUE,
  name = 'PDF Download',
  type = 'EVENT',
  eventDetails = list(
    useEventValue = TRUE,
    eventConditions = list(
      list(
        type = 'CATEGORY',
        matchType = 'EXACT',
        expression = 'PDF Download'
      ),
      list(
        type = 'LABEL',
        matchType = 'EXACT',
        expression = 'January brochure'
      )
    )
  )
)

## Create a Goal object based on a number of pages visited in a session:
Goal <- list(
  id = '10',
  active = TRUE,
  name = 'Visited more than 3 pages',
  type = 'VISIT_NUM_PAGES',
  visitNumPagesDetails = list(
    comparisonType = 'GREATER_THAN',
    comparisonValue = 3
  )
)

## Create a Goal object based on the number of seconds spent on the site
Goal <- list(
  id = '11',
  active = TRUE,
  name = 'Stayed for more than 2 minutes',
  type = 'VISIT_TIME_ON_SITE',
  visitTimeOnSiteDetails = list(
    comparisonType = 'GREATER_THAN',
    comparisonValue = 120
  )
)
ga_goal_list

List goals

Description
List goals

Usage
ga_goal_list(accountId, webPropertyId, profileId)

Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>profileId</td>
<td>Profile Id</td>
</tr>
</tbody>
</table>

Value
Goal list

See Also
Other goal management functions: ga_goal_add(), ga_goal_update(), ga_goal()

ga_goal_update

Updates an existing goal.

Description
Updates an existing goal.

Usage

```r
ga_goal_update(
  Goal,
  accountId,
  webPropertyId,
  viewId,
  goalId,
  method = c("PUT", "PATCH")
)
```
Arguments

Goal The Goal object to be updated See examples from ga_goal_add()
accountId Account Id of the account in which to modify the Goal
webPropertyId Property Id of the property in which to modify the Goal
viewId View Id of the view in which to modify the Goal
goalId The id of the goal to be modified
method PUT by default. For patch semantics use PATCH

Value

A goalManagement object

See Also

https://developers.google.com/analytics/devguides/config/mgmt/v3/mgmtReference/#Goals
Other goal management functions: ga_goal_add(), ga_goal_list(), ga_goal()

Examples

## Not run:

```r
# Change the goal 11 to visits over 3 minutes
Goal <- list(
  active = TRUE,
  name = 'Stayed for more than 3 minutes',
  type = 'VISIT_TIME_ON_SITE',
  visitTimeOnSiteDetails = list(
    comparisonType = 'GREATER_THAN',
    comparisonValue = 180
  )
)
ga_goal_update(Goal, accountId, propertyId, viewId, 11)
```

```r
# Change destination url for goal 17
Goal <- list(
  urlDestinationDetails = list(
    url = '\\checkout\\success'
  )
)
```

```
# Only the fields we're changing required because we're using PATCH method
ga_goal_update(Goal, accountId, propertyId, viewId, 17, method = "PATCH")
```

## End(Not run)
Get current dimensions and metrics available in GA API.

Description

Get current dimensions and metrics available in GA API.

Usage

```r
ga_meta(
  version = c("universal", "data"),
  propertyId = NULL,
  cached = TRUE,
  no_api = FALSE
)
```

Arguments

- `version`: The Google Analytics API metadata to fetch - "universal" for Universal and earlier versions, "data" for Google Analytics 4
- `propertyId`: If requesting from Google Analytics 4, pass the propertyId to get metadata specific to that property. Leaving it NULL or 0 will return universal metadata
- `cached`: Whether to use a cached version or to use the API to fetch the results again
- `no_api`: Don’t call the API, just return `googleAnalyticsR::meta4`

Value

dataframe of dimensions and metrics available to use

See Also

- [metadata/columns/list](https://developers.google.com/analytics/devguides/reporting/metadata/v3/reference/metadata/columns/list)
- [data/v1/rest/v1alpha/properties/getMetadata](https://developers.google.com/analytics/devguides/reporting/data/v1/rest/v1alpha/properties/getMetadata)

Examples

```r
## Not run:

# universal analytics
ga_meta()

# Google Analytics 4 metadata from the Data API
ga_meta("data")

# Google Analytics 4 metadata for a particular Web Property
ga_meta("data", propertyId = 206670707)
```
## ga_model

Use a model

**Description**

Use a model created by `ga_model_make`

**Usage**

```r
ga_model(viewId, model, load_libs = TRUE, ...)
```

**Arguments**

- `viewId`: The GA viewId to operate on
- `model`: A file location of a model object or a model object created by `ga_model_make`
- `load_libs`: Whether to load the library requirements into your namespace
- `...`: Other arguments to pass into the model as needed

**See Also**

Other GA modelling functions: `ga_model_edit()`, `ga_model_example()`, `ga_model_load()`, `ga_model_make()`, `ga_model_save()`, `ga_model_shiny_load()`, `ga_model_shiny_template()`, `ga_model_shiny()`, `ga_model_write()`

**Examples**

```r
# models that come with the package
ga_model_example()

## Not run:

# your own Google Analytics viewID
my_viewid <- 81416156

# load the model (equivalent to ga_model_load())
decomp_ga <- ga_model_example("decomp_ga.gamr")

# apply model to your data
d1 <- ga_model(my_viewid, model = decomp_ga)

# change default date range to 20 days ago to yesterday
d2 <- ga_model(my_viewid, model = decomp_ga,
               date_range = c("20daysAgo","yesterday"))
```
## ga_model_edit

Edit a created ga_model

### Description

Change features of a model by changing the functions within it.

### Usage

```r
ga_model_edit(
  model,
  data_f = NULL,
  required_columns = NULL,
  model_f = NULL,
  required_packages = NULL,
  description = NULL,
  outputShiny = shiny::plotOutput,
  renderShiny = shiny::renderPlot,
  inputShiny = NULL,
  output_f = NULL
)
```

### Arguments

- **model**: The model to edit - if a filepath will load model and save back edited model to the same file
- **data_f**: A function that gets the data
- **required_columns**: What dimensions and metrics are required
- **model_f**: A function that inputs data, and outputs a list of assets - must take data from result of `data_f` in first argument
- **required_packages**: The packages needed for `data_f` and `model_f` to work
- **description**: An optional description of what the model does
- **outputShiny**: A shiny UI output function that will display the results `renderShiny`
- **renderShiny**: A shiny render function that will create the output for `outputShiny` from `output_f`
- **inputShiny**: Optional input shiny functions (like `dateInput()`) that will be used within the model’s Shiny module. The id should be exactly the same as one of the variables in the model functions.
- **output_f**: A function that inputs the output from `model_f`, outputs a visualisation
See Also
Other GA modelling functions: \texttt{ga\_model\_example()}, \texttt{ga\_model\_load()}, \texttt{ga\_model\_make()}, \texttt{ga\_model\_save()}, \texttt{ga\_model\_shiny\_load()}, \texttt{ga\_model\_shiny\_template()}, \texttt{ga\_model\_shiny()}, \texttt{ga\_model\_write()}, \texttt{ga\_model()}.

Examples

\begin{verbatim}
## Not run:
decomp_ga <- ga_model_example("decomp_ga.gamr")
decomp_ga

# edit its description
ga_model_edit(decomp_ga, description = "Changed")

## End(Not run)
\end{verbatim}

\begin{verbatim}
# example .gamr files included with the package
ga_model_example()

# load one example
ga_model_example("ga4-trend.gamr")
\end{verbatim}
**ga_model_load**  
*Load a created model*

**Description**
Load a created model

**Usage**
```r
ga_model_load(filename = "my-model.gamr")
```

**Arguments**
- **filename**: name to load model from

**See Also**
Other GA modelling functions: `ga_model_edit()`, `ga_model_example()`, `ga_model_make()`, `ga_model_save()`, `ga_model_shiny_load()`, `ga_model_shiny_template()`, `ga_model_shiny()`, `ga_model_write()`, `ga_model()`

**Examples**
```r
# models used in ga_model_example() are here:
location <- system.file("models","examples","decomp_ga.gamr",package = "googleAnalyticsR")

ga_model_load(location)
```

---

**ga_model_make**  
*Modelling function factory for Google Analytics data*

**Description**
Create `ga_model` objects for easy application of models to data

**Usage**
```r
ga_model_make(
  data_f, 
  required_columns, 
  model_f, 
  output_f = function(df, ...) { plot(df) }, 
  required_packages = NULL, 
  description = NULL,
```

---
outputShiny = shiny::plotOutput,
renderShiny = shiny::renderPlot,
inputShiny = shiny::tagList()
)

Arguments

data_f A function that gets the data
required_columns What dimensions and metrics are required
model_f A function that inputs data, and outputs a list of assets - must take data from result of data_f in first argument
output_f A function that inputs the output from model_f, outputs a visualisation
required_packages The packages needed for data_f and model_f to work
description An optional description of what the model does
outputShiny A shiny UI output function that will display the results renderShiny
renderShiny A shiny render function that will create the output for outputShiny from output_f
inputShiny Optional input shiny functions (like dateInput()) that will be used within the model’s Shiny module. The id should be exactly the same as one of the variables in the model functions.

Details

The passed functions should all have ... to make them flexible in what arguments can be added. Do not have the same argument names in both functions. The data_f function result will feed to model_f

Value

A ga_model object to pass to ga_model

See Also

Other GA modelling functions: ga_model_edit(), ga_model_example().ga_model_load().ga_model_save(), ga_model_shiny_load(), ga_model_shiny_template(), ga_model_shiny().ga_model_write(), ga_model()

Examples

## Not run:

get_model_data <- function(viewId,
    date_range = c(Sys.Date()- 300, Sys.Date()),
    ...){
google_analytics(viewId,
    date_range = date_range,
ga_model_refresh

```r
decompose_sessions <- function(df, ...){
  decompose(ts(df$sessions, frequency = 7))
}

decomp_ga <- ga_model_make(get_model_data,
                           required_columns = c("date", "sessions"),
                           model_f = decompose_sessions,
                           description = "Performs decomposition and creates plot")

# fetches data and outputs decomposition
ga_model(81416156, decomp_ga)

# save the model for later
model_location <- "decomp_ga.gamr"
ga_model_save(decomp_ga, filename = model_location)

# can load model from file
ga_model(81416156, model_location)

# or load model to an object and use
model2 <- ga_model_load(model_location)
ga_model(81416156, model2)

# for shiny include functions for the UI and server rendering
decomp_ga <- ga_model_make(get_model_data,
                           required_columns = c("date", "sessions"),
                           model_f = decompose_sessions,
                           output_f = function(df, ...)(graphics::plot(df)),
                           description = "Performs decomposition and creates a plot",
                           outputShiny = shiny::plotOutput,
                           renderShiny = shiny::renderPlot)

## End(Not run)
```

ga_model_refresh

Refresh a model

Description

Sometimes necessary if functions were created under differing package versions

Usage

```r
ga_model_refresh(model)
```
Arguments

model Model or file location of model .gamr file

Examples

```r
## Not run:
decomp_ga <- ga_model_example("decomp_ga.gamr")
decomp_ga <- ga_model_refresh(decomp_ga)

## End(Not run)
```

---

**ga_model_save**

Save a created model

**Usage**

```r
ga_model_save(model, filename = "my-model.gamr")
```

**Arguments**

- model model to save
- filename name to save model under

**See Also**

Other GA modelling functions: `ga_model_edit()`, `ga_model_example()`, `ga_model_load()`, `ga_model_make()`, `ga_model_shiny_load()`, `ga_model_shiny_template()`, `ga_model_shiny()`, `ga_model_write()`, `ga_model()`

**Examples**

```r
## Not run:
# load the model (equivalent to ga_model_load())
decomp_ga <- ga_model_example("decomp_ga.gamr")

# save it somewhere else
ga_model_save(decomp_ga, "somewhereelse.gamr")

## End(Not run)
```
Create a Shiny app from a ga_model file

Usage

```r
ga_model_shiny(
  models,
  template = ga_model_shiny_template("basic"),
  header_boilerplate = TRUE,
  title = "ga_model_shiny",
  auth_dropdown = c("ga4", "universal", "none"),
  web_json = Sys.getenv("GAR_CLIENT_WEB_JSON"),
  date_range = TRUE,
  scopes = "https://www.googleapis.com/auth/analytics.readonly",
  deployed_url = "",
  local_folder = "",
  ...
)
```

Arguments

- **models**: The ga_model file location ("my_model.gamr") or a ga_model object - can pass in multiple as a list
- **template**: The template Shiny files for the Shiny app - passed to shiny::runApp()
- **header_boilerplate**: Whether to add header boilerplate to the template
- **title**: The title of the Shiny app
- **auth_dropdown**: What type of account picker to include
- **web_json**: The client.id json file for Web
- **date_range**: Most templates support a date_range global input for the data import functions, set this to FALSE to remove it
- **scopes**: The scope the API requests will be under
- **deployed_url**: If deploying Shiny app to a server, put the URL of the deployed app here so the authentication will redirect to the correct place
- **local_folder**: If not empty, will not launch Shiny app but write code to the folder location you put here

... Extra macro variables the template may support: a named list with the name being a template variable
Details

As `ga_model` objects have standardised code, they can be used to build standard templated Shiny apps. Templates are made using the `whisker.render` function.

Some templates are included with the package, seen via `ga_model_shiny_template("list")`. Templates hold macro variables indicated via `{{ macro_name }}` in the Shiny app template code. See `ga_model_shiny_template("basic_app",TRUE)` for an example showing a minimal viable app. Templates can be files such as `ui.R` or `app.R` files; folders containing `ui.R` and `app.R` files; or `ui.R` with `html` files for advanced themes - see [Shiny HTML templates](#). All additional files that may be in the folder are also copied over (such as `global.R` or `www/` folders).

Templates contain code to allow multi-user login via Google OAuth2.

If your template is pointing at a file such as `ui.R` or `app.R` it will create an `app.R` Shiny object. If your template is pointing at a directory it will check for the presence of `ui.R` within the folder. In either case if the `server.R` is missing it will use the boilerplate version from `ga_model_shiny_template("boilerplate")`.

By default the Shiny app is launched which in most cases will prompt authorisation for your Google Analytics. You can instead write the app out using `local_folder` to a valid location for deployment later.

Template macro variables

- `{{{ model_libraries }}}`- Adds `library()` calls based on `models$required_packages`
- `{{{ web_json }}}`- Adds Google OAuth2 client for web applications
- `{{{ scopes }}}`- Adds Google OAuth2 scopes for the API calls
- `{{{ deployed_url }}}`- Adds option(`googleAuthR.redirect`) option for deployed Shiny apps
- `{{{ model_load }}}`- Adds `ga_model_load` calls loading all models in the list passed to this function's `models` argument. It creates R objects called 'model1', 'model2' etc. in the Shiny app code
- `{{{ model_list }}}`- Adds a list of the model objects after `model_load`. Useful for creating custom functions in themes that can loop over model objects
- `{{{ shiny_title }}}`- Adds the title to the Shiny app
- `{{{ auth_ui }}}`- Adds the correct dropdown Shiny module for picking a GA4 or Universal Analytics properties
- `{{{ date_range }}}`- Adds a shiny::`dateInput()` date selector with id "date_range" for use in model's data fetching functions
- `{{{ model_ui }}}`- Adds the models UI elements as configured in the `ga_model` object. It uses the object loaded above via the `model_load` macro. It looks like `model1$ui('model1')` in the code.
- `{{{ auth_server }}}`- Adds the authentication module's server side function
- `{{{ auth_accounts }}}`- Adds a call to `ga_account_list` for the appropriate GA account type (GA4 or Universal)
- `{{{ model_server }}}`- Adds the server side module for the models as configured in the `ga_model` configuration. It uses the object loaded above via the `model_load` macro. It looks like `model1$server('model1')` in the code.
• `{{{ model1 }}}` - Alternative to `model_load`, this will load the model file location instead, which you can pass to `ga_model_load()` in the template. `model1` is the first model passed, `model2` the second, etc.
• `{{{ your_argument }}}` - You can pass in your own custom variables to the template via the `...` argument of this function if they are named the same as the template macro variable

See Also

Other GA modelling functions: `ga_model_edit()`, `ga_model_example()`, `ga_model_load()`, `ga_model_make()`, `ga_model_save()`, `ga_model_shiny_load()`, `ga_model_shiny_template()`, `ga_model_write()`, `ga_model()`

Examples

```r
# see Shiny templates included with the package
ga_model_shiny_template("list")

# see an example of an ui.R template with macros
ga_model_shiny_template("basic/ui.R", read_lines = TRUE)

# see an example of an app.R template with macros
ga_model_shiny_template("basic_app/app.R", read_lines = TRUE)

## Not run:
# a universal analytics model using default template "basic"
ga_model_shiny(
  ga_model_example("decomp_ga.gamr"),
  auth_dropdown = "universal")

# a template from a directory holding an app.R file
ga_model_shiny(
  ga_model_example("decomp_ga.gamr"),
  auth_dropdown = "universal",
  template = ga_model_shiny_template("basic_app"))

# a template from only an ui.R file that will import boilerplate server.R
ga_model_shiny(
  ga_model_example("decomp_ga.gamr"),
  auth_dropdown = "universal",
  template = ga_model_shiny_template("basic/ui.R"))

# a template from a custom html based theme
ga_model_shiny(
  ga_model_example("decomp_ga.gamr"),
  auth_dropdown = "universal",
  template = ga_model_shiny_template("html_based"))

# a template using library(argonDash)
ga_model_shiny(
  ...)```
ga_model_example("ga-effect.gamr"),
  title = "Argon Demo",
  auth_dropdown = "universal",
  template = ga_model_shiny_template("argonDash") )

# multiple models
m3 <- ga_model_example("time-normalised.gamr")
m4 <- ga_model_example("ga-effect.gamr")

# launch in gentelella template
ga_model_shiny(list(m4, m3), auth_dropdown = "universal",
  template = ga_model_shiny_template("gentelella"))

# you can make custom ui embedded within the template file
# use {{ model_list }} to work with the models in the ui.R

# below adds custom macro 'theme' and a custom ui in box tabs
ga_model_shiny(list(m4, m3), auth_dropdown = "universal",
  template = ga_model_shiny_template("shinythemes"),
  theme = "yeti")

# shinydashboard's custom ui functions put a model in each side tab
ga_model_shiny(list(m4, m3), auth_dropdown = "universal",
  template = ga_model_shiny_template("shinydashboard"),
  skin = "green")

# send in lots of theme variables to bslib in shiny > 1.6.0
ga_model_shiny(list(m4, m3), auth_dropdown = "universal",
  template = ga_model_shiny_template("basic_bslib"),
  bg = "white", fg = "red", primary = "grey")

# write out an app to a local folder
ga_model_shiny(list(m4, m3), auth_dropdown = "universal",
  template = ga_model_shiny_template("basic_bslib"),
  bg = "white", fg = "red", primary = "grey",
  local_folder = "deploy_shiny")

## End(Not run)
Usage

```
  ga_model_shiny_load(model_n, ...)
```

Arguments

- `model_n` The templated name of a model e.g. 'model1'
- `...` Other arguments passed from shiny server

See Also

Other GA modelling functions: `ga_model_edit()`, `ga_model_example()`, `ga_model_load()`, `ga_model_make()`, `ga_model_save()`, `ga_model_shiny_template()`, `ga_model_shiny()`, `ga_model_write()`, `ga_model()`

---

Description

`ga_model_shiny_template` gets a pre-created template from the googleAnalyticsR samples

Usage

```
  ga_model_shiny_template(name = "list", read_lines = FALSE)
```

Arguments

- `name` the template name
- `read_lines` If TRUE will use `readLines()` to print out the template contents

See Also

Other GA modelling functions: `ga_model_edit()`, `ga_model_example()`, `ga_model_load()`, `ga_model_make()`, `ga_model_save()`, `ga_model_shiny_load()`, `ga_model_shiny()`, `ga_model_write()`, `ga_model()`
ga_model_write

Write the ga_model functions to a file

Description
Write the ga_model functions to a file

Usage

\[
\text{ga_model_write(model, filepath = "ga_model.R")}
\]

Arguments

- `model`: The ga_model object to extract functions from to write, or a filepath to a model
- `filepath`: The filepath to write the functions to

See Also

Other GA modelling functions: `ga_model_edit()`, `ga_model_example()`, `ga_model_load()`, `ga_model_make()`, `ga_model_save()`, `ga_model_shiny_load()`, `ga_model_shiny_template()`, `ga_model_shiny()`, `ga_model()`

Examples

```r
## Not run:

decomp_ga <- ga_model_example("decomp_ga.gamr")
ga_model_write(decomp_ga, "a_file.R")

## End(Not run)
```

ga_mp_cid

Generate a random client_id

Description
This has a random number plus a timestamp

Usage

\[
\text{ga_mp_cid(seed = NULL)}
\]

Arguments

- `seed`: If you set a seed, then the random number will be the same for each value
See Also

Other Measurement Protocol functions: `ga_mp_event_item()`, `ga_mp_event()`, `ga_mp_send()`

---

**ga_mp_event**

Create a Measurement Protocol Event

**Description**

[Experimental] This creates an event to send via `ga_mp_send`

**Usage**

```r
ga_mp_event(name, params = NULL, items = NULL)
```

**Arguments**

- `name`: The event name to send in
- `params`: Optional event parameters sent in as a named list
- `items`: Optional items created via `ga_mp_event_item`

**See Also**

Other Measurement Protocol functions: `ga_mp_cid()`, `ga_mp_event_item()`, `ga_mp_send()`

**Examples**

```r
ga_mp_event("custom_event")
ga_mp_event("custom_event", params = list(my_param = "SUPER"))
```

---

**ga_mp_event_item**

Create an Measurement Protocol Item Property for an Event

**Description**

[Experimental] Some events work with item properties
Usage

ga_mp_event_item(
    item_id = NULL,
    item_name = NULL,
    coupon = NULL,
    discount = NULL,
    affiliation = NULL,
    item_brand = NULL,
    item_category = NULL,
    item_variant = NULL,
    price = NULL,
    currency = NULL
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_id</td>
<td>Item ID</td>
</tr>
<tr>
<td>item_name</td>
<td>Item Name</td>
</tr>
<tr>
<td>coupon</td>
<td>Coupon</td>
</tr>
<tr>
<td>discount</td>
<td>Discount</td>
</tr>
<tr>
<td>affiliation</td>
<td>Affiliation</td>
</tr>
<tr>
<td>item_brand</td>
<td>Brand</td>
</tr>
<tr>
<td>item_category</td>
<td>Category</td>
</tr>
<tr>
<td>item_variant</td>
<td>Variant</td>
</tr>
<tr>
<td>price</td>
<td>Price</td>
</tr>
<tr>
<td>currency</td>
<td>Currency</td>
</tr>
</tbody>
</table>

See Also

Other Measurement Protocol functions: `ga_mp_cid()`, `ga_mp_event()`, `ga_mp_send()`

Examples

# one item
ga_mp_event_item(item_name = "jeggings",
                  price = 8.88,
                  item_variant = "Black")

# many items in a list
items <- list(
    ga_mp_event_item(item_id = "SKU_12345",
                      price = 9.99,
                      item_brand = "Gucci"),
    ga_mp_event_item(item_name = "jeggings",
                      price = 8.88,
                      item_variant = "Black"))
# construct an event with its own fields
ga_mp_event("add_payment_info",
    params = list(coupon = "SUMMER_FUN",
                  payment_type = "Credit Card",
                  value = 7.77,
                  currency = "USD"),
    items = items)

---

**ga_mp_send**

*Make a Measurement Protocol v2 request*

## Description

[Experimental] Create a server side call to Google Analytics 4 via its Measurement Protocol

Use `ga_mp_connection` to set up the Measurement Protocol connections to pass to `ga_mp_send`. If using Google Tag Manager Server-Side, you can also set up a custom endpoint.

## Usage

```r
ga_mp_send(
    events,
    client_id,
    connection,
    user_id = NULL,
    debug_call = FALSE,
    timestamp_micros = NULL,
    user_properties = NULL,
    non_personalized_ads = TRUE
)
```

```r
ga_mp_connection(
    measurement_id,
    api_secret = Sys.getenv("MP_SECRET"),
    endpoint = NULL,
    preview_header = NULL
)
```

## Arguments

- `events`  The events to send
- `client_id`  The client_id to associate with the event
- `connection`  The connection details created by `ga_mp_connection`
- `user_id`  Optional. Unique id for the user
- `debug_call`  Send hits to the Google debug endpoint to validate hits.
- `timestamp_micros`  Optional. A Unix timestamp (in microseconds) for the time to associate with the event.
user_properties
Optional. The user properties for the measurement sent in as a named list.

non_personalized_ads
Optional. Set to true to indicate these events should not be used for personalized ads.

measurement_id
The measurement ID associated with a stream

api_secret
The secret generated in the GA4 UI - by default will look for environment arg MP_SECRET

endpoint
If NULL will use Google default, otherwise set to the URL of your Measurement Protocol custom endpoint

preview_header
Only needed for custom endpoints. The X-Gtm-Server-Preview HTTP Header found in your GTM debugger

Details
Create an API secret via Admin > Data Streams > choose your stream > Measurement Protocol > Create
To see event parameters, create custom fields in your GA4 account first, to see them in your reports 24hrs after you send them in with this function via Custom definitions > Create custom dimensions - dimension name will be how it looks like in the reports, event parameter will be the parameter you have sent in with the event.
user_id can be used for cross-platform analysis
timestamp_micros should only be set to record events that happened in the past. This value can be overridden via user_property or event timestamps. Events can be backdated up to 48 hours. Note microseconds, not milliseconds.
user_properties - describe segments of your user base, such as language preference or geographic location. See User properties
Ensure you also have user permission as specified in the feature policy
Invalid events are silently rejected with a 204 response, so use debug_call=TRUE to validate your events first.

Value
TRUE if successful, if debug_call=TRUE then validation messages if not a valid hit.

See Also
Measurement Protocol (Google Analytics 4)
Other Measurement Protocol functions: ga_mp_cid(), ga_mp_event_item(), ga_mp_event()

Examples
# preferably set this in .Renviron
Sys.setenv(MP_SECRET="MY_SECRET")

# your GA4 settings
my_measurement_id <- "G-1234"

my_connection <- ga_mp_connection(my_measurement_id)

a_client_id <- 123.456
event <- ga_mp_event("an_event")
ga_mp_send(event, a_client_id, my_connection, debug_call = TRUE)

# multiple events at same time in a batch
another <- ga_mp_event("another_event")
ga_mp_send(list(event, another),
a_client_id,
my_connection,
debup_call = TRUE)

## Not run:
# you can see sent events in the real-time reports
my_property_id <- 206670707
ga_data(my_property_id,
dimensions = "eventName",
metrics = "eventCount",
dim_filters = ga_data_filter(
  eventName == c("an_event","another_event")),
realtime = TRUE)

## End(Not run)

# custom GTM server side endpoint
my_custom_connection <- ga_mp_connection(
  my_measurement_id,
derpoint = "https://gtm.example.com",
preview_header = "ZW52LTV80WdPOExNWFlYjA0Njk4NmQ="
)

---

ga_remarketing_build  Create a remarketing audience for creation

Description

Create definitions to be used within ga_remarketing_create

Usage

ga_remarketing_build(
  segment,
  membershipDurationDays = NULL,
daysToLookBack = NULL,
state_duration = c("TEMPORARY", "PERMANENT")
)
Arguments

- **segment**: The definition of the segment (v3 syntax)
- **membershipDurationDays**: Number of days (in the range 1 to 540) a user remains in the audience.
- **daysToLookBack**: The look-back window lets you specify a time frame for evaluating the behavior that qualifies users for your audience.
- **state_duration**: If to be used in a state based audience, whether to make the segment temporary or permanent.

Details

The look-back window lets you specify a time frame for evaluating the behavior that qualifies users for your audience. For example, if your filters include users from Central Asia, and Transactions Greater than 2, and you set the look-back window to 14 days, then any user from Central Asia whose cumulative transactions exceed 2 during the last 14 days is added to the audience.

See Also

Other remarketing management functions: `ga_remarketing_create()`, `ga_remarketing_estimate()`, `ga_remarketing_get()`, `ga_remarketing_list()`

Examples

```r
## Not run:
adword_list <- ga_adwords_list(123456, "UA-123456-1")

adword_link <- ga_adword(adword_list$id[[1]])

segment_list <- ga_segment_list()$items$definition

my_remarketing1 <- ga_remarketing_build(segment_list[[1]],
    state_duration = "TEMPORARY",
    membershipDurationDays = 90,
    daysToLookBack = 14)

my_remarketing2 <- ga_remarketing_build(segment_list[[2]],
    state_duration = "PERMANENT",
    membershipDurationDays = 7,
    daysToLookBack = 31)

# state based only can include exclusions
ga_remarketing_create(adwords_link = adword_link,
    include = my_remarketing1,
    exclude = my_remarketing2,
    audienceType = "STATE_BASED",
    name = "my_remarketing_seg1")

## End(Not run)
```
Create a new remarketing audience

Description

Create a remarketing audiences built via \texttt{ga_remarketing_build}

Usage

\begin{verbatim}
ga_remarketing_create(
  adwordsLinkId,
  include,
  exclude = NULL,
  audienceType = c("SIMPLE", "STATE_BASED"),
  name = NULL
)
\end{verbatim}

Arguments

- \texttt{adwordsLinkId}: The adwords link to add the remarketing audience to
- \texttt{include}: A \texttt{ga4_remarketing_segment} object to include via \texttt{ga_remarketing_build}
- \texttt{exclude}: If \texttt{audienceType="STATE_BASED"}, a \texttt{ga4_remarketing_segment} object to exclude via \texttt{ga_remarketing_build}
- \texttt{audienceType}: SIMPLE or STATE_BASED
- \texttt{name}: An optional name, if not supplied one will be generated

Details

This builds and calls the API to create the remarketing audience based on the segments you have defined.

See Also

Other remarketing management functions: \texttt{ga_remarketing_build()}, \texttt{ga_remarketing_estimate()}, \texttt{ga_remarketing_get()}, \texttt{ga_remarketing_list()}

Examples

\begin{verbatim}
## Not run:
adword_list <- ga_adwords_list(123456, "UA-123456-1")
adword_link <- ga_adword(adword_list$id[[1]])
segment_list <- ga_segment_list()$items$definition
my_remarketing1 <- ga_remarketing_build(segment_list[[1]],
\end{verbatim}
ga_remarketing_estimate

Estimate number of users added to the segment yesterday

Description

Estimate number of users added to the segment yesterday

Usage

ga_remarketing_estimate(remarketingAudience)

Arguments

remarketingAudience

A remarketing audience object from ga_remarketing_get
Takes the segment definition from a remarketing audiences and runs it against
the viewId to see current estimated users
The total audience size is this figure for every membershipDurationDay from
yesterday

Value

data.frame

See Also

About remarketing audiences
Other remarketing management functions: ga_remarketing_build(), ga_remarketing_create(),
ga_remarketing_get(), ga_remarketing_list()
**ga_remarketing_get**  
*Get a remarketing audience*

**Description**
Get a remarketing audience

**Usage**
```
ga_remarketing_get(accountId, webPropertyId, remarketingAudienceId)
```

**Arguments**
- `accountId`  
  Account Id
- `webPropertyId`  
  Web Property Id
- `remarketingAudienceId`  
  The ID of the remarketing audience to retrieve.

**Value**
Remarketing Audience object

**See Also**
- About remarketing audiences
- Other remarketing management functions: `ga_remarketing_build()`, `ga_remarketing_create()`, `ga_remarketing_estimate()`, `ga_remarketing_list()`

---

**ga_remarketing_list**  
*List remarketing audiences*

**Description**
List remarketing audiences

**Usage**
```
ga_remarketing_list(accountId, webPropertyId)
```

**Arguments**
- `accountId`  
  Account Id
- `webPropertyId`  
  Web Property Id
**Value**

Remarking audience list

**See Also**

About remarking audiences

Other remarking management functions: `ga_remarketing_build()`, `ga_remarketing_create()`, `ga_remarketing_estimate()`, `ga_remarketing_get()`

---

**ga_segment_list**

*Get segments user has access to*

**Description**

Get segments user has access to

**Usage**

`ga_segment_list()`

**Value**

Segment list

**See Also**

Other management API functions: `ga_experiment_list()`, `ga_experiment()`, `ga_filter_add()`, `ga_filter_apply_to_view()`, `ga_filter_update_filter_link()`, `ga_filter_update()`

---

**ga_trackme**

*Opt in or out of googleAnalyticsR usage tracking*

**Description**

You can opt-in or out to sending a measurement protocol hit when you load the package for use in the package’s statistics via this function. No personal data is collected.

If you opt in, `ga_trackme_event()` is the function that fires. You can use `debug_call=TRUE` to see what would be sent before opting in or out.

**Usage**

`ga_trackme()`

`ga_trackme_event(debug_call = FALSE, say_hello = NULL)`
Arguments

debug_call    Set as a debug event to see what would be sent
say_hello     If you want to add your own custom message to the event sent, add it here!

Details

Running `ga_trackme_event()` function will send a Measurement Protocol hit via `ga_mp_send`
only if the `~/.R/optin-googleanalyticsr` file is present

Examples

```r
# control your tracking choices via a menu if in interactive session
if(interactive()){
  ga_trackme()
}

# this only works with a valid opt-in file present
ga_trackme_event()

# see what data is sent
ga_trackme_event(debug_call=TRUE)

# add your own message!
ga_trackme_event(debug_call = TRUE, say_hello = "err hello Mark")
```

---

**ga_unsampled**

*Get Unsampled Report Meta Data*

Description

Get Unsampled Report Meta Data

Usage

`ga_unsampled(accountId, webPropertyId, profileId, unsampledReportId)`

Arguments

- `accountId`    Account Id
- `webPropertyId`    Web Property Id
- `profileId`    Profile Id
- `unsampledReportId`    Unsampled Report Id

Value

Unsampled Report Meta Data
ga_unsampled_download

Download Unsampled Report from Google Drive. You must be authenticated with the same account that you setup the unsampled report. This means service account authentication is not supported.

Description

Download Unsampled Report from Google Drive. You must be authenticated with the same account that you setup the unsampled report. This means service account authentication is not supported.

Usage

```r
ga_unsampled_download(
  reportTitle,
  accountId,
  webPropertyId,
  profileId,
  downloadFile = TRUE
)
```

Arguments

- `reportTitle`: Title of Unsampled Report (case-sensitive)
- `accountId`: Account Id
- `webPropertyId`: Web Property Id
- `profileId`: Profile Id
- `downloadFile`: Default TRUE, whether to download, if FALSE returns a dataframe instead

Value

- file location if `downloadFile` is TRUE, else a `data.frame` of download

See Also

Other unsampled download functions: `ga_unsampled_list()`, `ga_unsampled()`
Examples

```r
## Not run:

# get data.frame of unsampled reports you have available
unsample_list <- ga_unsampled_list(accountId = "12345",
             webPropertyId = "UA-12345-4",
             profileId = "129371234")

# loop through unsampled reports and download as a list of data.frames
dl <- lapply(unsample_list$title, ga_unsampled_download,
             accountId = "12345",
             webPropertyId = "UA-12345-4",
             profileId = "129371234",
             downloadFile = FALSE)

# inspect first data.frame
dl[[1]]

# download unsampled report to csv file
ga_unsampled_download("my_report_title",
             accountId = "12345",
             webPropertyId = "UA-12345-4",
             profileId = "129371234")

## End(Not run)
```

---

ga_unsampled_list  List Unsampled Reports

Description

List Unsampled Reports

Usage

```r
ga_unsampled_list(accountId, webPropertyId, profileId)
```

Arguments

- accountId: Account Id
- webPropertyId: Web Property Id
- profileId: Profile Id

Value

Unsampled Reports List
See Also

Other unsampled download functions: `ga_unsampled_download()`, `ga_unsampled()`

Examples

```r
## Not run:

# get data.frame of unsampled reports you have available
unsample_list <- ga_unsampled_list(accountId = "12345",
                                   webPropertyId = "UA-12345-4",
                                   profileId = "129371234")

# loop through unsampled reports and download as a list of data.frames
dl <- lapply(unsample_list$title, ga_unsampled_download,
              accountId = "12345",
              webPropertyId = "UA-12345-4",
              profileId = "129371234",
              downloadFile = FALSE)

# inspect first data.frame
dl[[1]]

# download unsampled report to csv file
ga_unsampled_download("my_report_title",
                     accountId = "12345",
                     webPropertyId = "UA-12345-4",
                     profileId = "129371234")

## End(Not run)
```

---

**Description**

If you supply more than one email, then batch processing will be applied. Batching has special rules that give you 30 operations for the cost of one API call against your quota. When batching you will only get a TRUE result on successful batch, but individual entries may have failed. Check via `ga_users_list` afterwards and try to add individual linkIds to get more descriptive error messages.

**Usage**

```r
ga_users_add(
    email,
```
ga_users_delete

```
permissions,
accountId,
webPropertyId = NULL,
viewId = NULL
)
```

**Arguments**

- **email**: The email(s) of the user(s) to add. Has to have a Google account.
- **permissions**: Which permissions to add as a vector - "MANAGE_USERS", "EDIT", "COLLABORATE", "READ_AND_ANALYZE"
- **accountId**: Account Id
- **webPropertyId**: Web Property Id - set to NULL to operate on account level only
- **viewId**: ViewId - set to NULL to operate on webProperty level only

**Value**

TRUE if successful

**See Also**

- Google help article on user permissions
- Other User management functions: `ga_users_delete_linkid()`, `ga_users_delete()`, `ga_users_list()`, `ga_users_update()`

**Examples**

```r
## Not run:
library(googleAnalyticsR)
ga_auth()
ga_users_add(c("the_email@company.com", "another_email@company.com"),
permissions = "EDIT", accountId = 47480439)
```

## End(Not run)

---

**ga_users_delete**

Delete all user access for an email

**Description**

This is a wrapper around calls to `ga_users_list` and `ga_users_delete_linkid`. If you want more fine-grained control look at those functions.

The user email is deleted from all web properties and views underneath the accountId you provide.
Usage

\texttt{ga_users_delete(email, accountId)}

Arguments

\begin{itemize}
\item \texttt{email} \hspace{0.5cm} The email of the user to delete
\item \texttt{accountId} \hspace{0.5cm} The accountId that the user will be deleted from including all web properties and Views underneath.
\end{itemize}

Details

This deletes a user via their email reference for all webproperties and views for the account given.

See Also

\begin{itemize}
\item \texttt{Google Documentation}
\item Other User management functions: \texttt{ga_users_add()}, \texttt{ga_users_delete_linkid()}, \texttt{ga_users_list()}, \texttt{ga_users_update()}
\end{itemize}

Examples

\begin{verbatim}
## Not run:
library(googleAnalyticsR)
ga_auth()
ga_users_delete("brian@agency.com", 12345678)

# multiple emails
ga_users_delete(c("brian@agency.com", "bill@benland.com"), 12345678)

## End(Not run)
\end{verbatim}

---

\texttt{ga_users_delete_linkid}

\textit{Delete users access from account, webproperty or view level}

Description

The \texttt{linkId} is in the form of the accountId/webPropertyId/viewId colon separated from a link unique Id.

Delete user access by supplying the \texttt{linkId} for that user at the level they have been given access. It won’t work to delete user links at account level if they have been assigned at web property or view level - you will need to get the \texttt{linkId} for that level instead. e.g. a user needs \texttt{permissions.local} to be non-NULL to be deleted at that level. The parameter \texttt{check} will do this check before deletion and throw an error if they can not be deleted. Set this to \texttt{check=FALSE} to suppress this behaviour.
If you supply more than one linkId, then batch processing will be applied. Batch processing has special rules that give you 30 operations for the cost of one API call against your quota. When batching you will only get a TRUE result on successful batch, but individual linkIds may have failed. Check via ga_users_list afterwards and try to delete individual linkIds to get more descriptive error messages.

Usage

```r
ga_users_delete_linkid(
  linkId,
  accountId,
  webPropertyId = NULL,
  viewId = NULL,
  check = TRUE
)
```

Arguments

- **linkId**: The linkId(s) that is available using ga_users_list e.g. 47480439:104185380183364788718
- **accountId**: Account Id
- **webPropertyId**: Web Property Id - set to NULL to operate on account level only
- **viewId**: viewId - set to NULL to operate on webProperty level only
- **check**: If the default TRUE will check that the user has user access at the level you are trying to delete them from - if not will throw an error.

Value

TRUE if the deletion is successful, an error if not.

See Also

- Google Documentation
- Other User management functions: ga_users_add(), ga_users_delete(), ga_users_list(), ga_users_update()

Examples

```r
## Not run:
library(googleAnalyticsR)
ga_auth()

# get the linkId for the user you want to delete
ga_users_list(47480439, webPropertyId = "UA-47480439-2", viewId = 81416156)
ga_users_delete_linkid("81416156:114834495587136933146",
  accountId = 47480439,
  webPropertyId = "UA-47480439-2",
  viewId = 81416156)
```
# check its gone
ga_users_list(47480439, webPropertyId = "UA-47480439-2", viewId = 81416156)

# can only delete at level user has access, the above deletion would have failed if via:
ga_users_delete_linkid("47480439:114834495587136933146", 47480439)

## End(Not run)

---

**ga_users_list**  
**List Users**

### Description
Get a list of Account level user links, or if you supply the webPropertyId or viewId it will show user
links at that level

### Usage

ga_users_list(accountId, webPropertyId = "~all", viewId = "~all")

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id - set to NULL to operate on account level only</td>
</tr>
<tr>
<td>viewId</td>
<td>viewId - set to NULL to operate on webProperty level only</td>
</tr>
</tbody>
</table>

### Details

Will list users on an account, webproperty or view level

### Value

A data.frame of user entity links including the linkId, email and permissions

### See Also

Account User Links Google Documentation

Other User management functions: `ga_users_add()`, `ga_users_delete_linkid()`, `ga_users_delete()`, `ga_users_update()`
Examples

## Not run:

library(googleAnalyticsR)
ga_auth()
ga_users_list(47480439)
ga_users_list(47480439, webPropertyId = "UA-47480439-2")
ga_users_list(47480439, webPropertyId = "UA-47480439-2", viewId = 81416156)

# use NULL to only list linkids for that level
ga_users_list(47480439, webPropertyId = NULL, viewId = NULL)

## End(Not run)

---

**ga_users_update**  
*Update a user access in Google Analytics*

**Description**

This is for altering existing user access.

**Usage**

```r
ga_users_update(
  linkId,
  update_object,
  accountId,
  webPropertyId = NULL,
  viewId = NULL
)
```

**Arguments**

- `linkId`  
The linkId to update
- `update_object`  
A list that will be turned into JSON that represents the new configuration for this linkId
- `accountId`  
Account Id
- `webPropertyId`  
Web Property Id - set to NULL to operate on account level only
- `viewId`  
viewId - set to NULL to operate on webProperty level only

**Value**

The new user object that has been altered.
ga_view

Get single View (Profile)

Description

Gets meta-data for a particular View/Profile

Usage

ga_view(accountId, webPropertyId, profileId)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>Account Id</td>
</tr>
<tr>
<td>webPropertyId</td>
<td>Web Property Id</td>
</tr>
<tr>
<td>profileId</td>
<td>Profile (View) Id</td>
</tr>
</tbody>
</table>

Value

A list of the Views meta-data.

See Also

Other account structure functions: ga_account_list(), ga_accounts(), ga_view_list(), ga_webproperty_list(), ga_webproperty()
Examples

```r
## Not run:
library(googleAnalyticsR)
ga_auth()
ga_view(1058095, webPropertyId = "UA-1058095-1", profileId = 1855267)
## End(Not run)
```

**ga_view_list**

*List View (Profile)*

Description

This gets the meta data associated with the Google Analytics Views for a particular accountId and webPropertyId. If you want all viewId information for all accounts you have access to, use `ga_account_list` instead.

Usage

`ga_view_list(accountId, webPropertyId)`

Arguments

- `accountId`: Account Id
- `webPropertyId`: Web Property Id e.g. UA-12345-1

Value

A `data.frame` of meta-data for the views

See Also

Other account structure functions: `ga_account_list()`, `ga_accounts()`, `ga_view()`, `ga_webproperty_list()`, `ga_webproperty()`

Examples

```r
## Not run:
library(googleAnalyticsR)
ga_auth()
views <- ga_view_list(1058095, "UA-1058095-1")
## End(Not run)
```
ga_webproperty

Get a web property

Description

Gets metadata for one particular web property.

Usage

ga_webproperty(accountId, webPropertyId)

Arguments

accountId  Account Id
webPropertyId  Web Property Id e.g. UA-12345-1

Value

webproperty

See Also

Other account structure functions: ga_account_list(), ga_accounts(), ga_view_list(), ga_view(),
googleAnalyticsR

Examples

## Not run:
library(googleAnalyticsR)
ga_auth()
wp <- ga_webproperty(1058095, "UA-1058095-1")

## End(Not run)

googleAnalyticsR

List web properties

Description

This gets the meta data for web properties associated with a particular accountId. If you want all
information available to your user, use ga_account_list instead.

Usage

ga_webproperty_list(accountId)
Arguments

accountId    Account Id

Value

A data.frame of webproperty meta-data

See Also

Other account structure functions: \texttt{ga_account_list()}, \texttt{ga_accounts()}, \texttt{ga_view_list()}, \texttt{ga_view()}, \texttt{ga_webproperty()}

Examples

## Not run:
library(googleAnalyticsR)
ga_auth()
aa <- ga_accounts()
wp <- ga_webproperty_list(aa$id[1])
## End(Not run)

---

googleAnalyticsR \quad \textit{Library for getting Google Analytics data into R}

Description

\texttt{googleAnalyticsR}

Follow the online documentation here: \url{https://code.markedmondson.me/googleAnalyticsR/}

---

\texttt{google_analytics} \quad \textit{Get Google Analytics v4 data}

Description

Fetch Google Analytics data using the v4 API. For the v3 API use \texttt{google_analytics_3}, for GA4’s Data API use \texttt{ga_data}. See website help for lots of examples: \url{Google Analytics Reporting API v4 in R}
Usage

google_analytics(
    viewId,
    date_range = NULL,
    metrics = NULL,
    dimensions = NULL,
    dim_filters = NULL,
    met_filters = NULL,
    filtersExpression = NULL,
    order = NULL,
    segments = NULL,
    pivots = NULL,
    cohorts = NULL,
    max = 1000,
    samplingLevel = c("DEFAULT", "SMALL", "LARGE"),
    metricFormat = NULL,
    histogramBuckets = NULL,
    anti_sample = FALSE,
    anti_sample_batches = "auto",
    slow_fetch = FALSE,
    useResourceQuotas = NULL,
    rows_per_call = 10000L
)

google_analytics_4(...)

Arguments

viewId character or data of viewId to get.
date_range character or date vector of format c(start, end) or for two date ranges: c(start1, end1, start2, end2)
metrics Metric(s) to fetch as a character vector. You do not need to supply the "ga:" prefix. See meta for a list of dimensons and metrics the API supports. Also supports your own calculated metrics.
dimensions Dimension(s) to fetch as a character vector. You do not need to supply the "ga:" prefix. See meta for a list of dimensons and metrics the API supports.
dim_filters A filter_clause_ga4 wrapping dim_filter
met_filters A filter_clause_ga4 wrapping met_filter
filtersExpression A v3 API style simple filter string. Not used with other filters.
order An order_type object
segments List of segments as created by segment_ga4
pivots Pivots of the data as created by pivot_ga4
cohorts Cohorts created by make_cohort_group
max Maximum number of rows to fetch. Defaults at 1000. Use -1 to fetch all results. Ignored when anti_sample=TRUE.
google_analytics

samplingLevel  Sample level
metricFormat   If supplying calculated metrics, specify the metric type
histogramBuckets  For numeric dimensions such as hour, a list of buckets of data.
anti_sample   If TRUE will split up the call to avoid sampling.
anti_sample_batches  “auto” default, or set to number of days per batch. 1 = daily.
slow_fetch   For large, complicated API requests this bypasses some API hacks that may result in 500 errors. For smaller queries, leave this as FALSE for quicker data fetching.
useResourceQuotas  If using GA360, access increased sampling limits. Default NULL, set to TRUE or FALSE if you have access to this feature.
rows_per_call   Set how many rows are requested by the API per call, up to a maximum of 100000.
...
Arguments passed to google_analytics

Value
A Google Analytics data.frame, with attributes showing row totals, sampling etc.

Row requests
By default the API call will use v4 batching that splits requests into 5 separate calls of 10k rows each. This can go up to 100k, so this means up to 500k rows can be fetched per API call, however the API servers will fail with a 500 error if the query is too complicated as the processing time at Google’s end gets too long. In this case, you may want to tweak the rows_per_call argument downwards, or fall back to using slow_fetch = FALSE which will send an API request one at a time. If fetching data via scheduled scripts this is recommended as the default.

Anti-sampling
anti_sample being TRUE ignores max as the API call is split over days to mitigate the sampling session limit, in which case a row limit won’t work. Take the top rows of the result yourself instead e.g. head(ga_data_unsampled,50300)
anti_sample being TRUE will also set samplingLevel='LARGE' to minimise the number of calls.

Resource Quotas
If you are on GA360 and have access to resource quotas, set the useResourceQuotas=TRUE and set the Google Cloud client ID to the project that has resource quotas activated, via gar_set_client or options.

Caching
By default local caching is turned on for v4 API requests. This means that making the same request as one this session will read from memory and not make an API call. You can also set the cache to disk via the ga_cache_call function. This can be useful when running RMarkdown reports using data.
Metrics

Metrics support calculated metrics like ga:users / ga:sessions if you supply them in a named vector. You must supply the correct 'ga:' prefix unlike normal metrics. You can mix calculated and normal metrics like so:

```r
customMetric <- c(sessionPerVisitor = "ga:sessions / ga:visitors","bounceRate","entrances")
```

You can also optionally supply a `metricFormat` parameter that must be the same length as the metrics. `metricFormat` can be: METRIC_TYPE_UNSPECIFIED, INTEGER, FLOAT, CURRENcy, PERCENT, TIME

All metrics are currently parsed to as.numeric when in R.

Dimensions

Supply a character vector of dimensions, with or without ga: prefix. Optionally for numeric dimension types such as ga:hour, ga:browserVersion, ga:sessionsToTransaction, etc. supply histogram buckets suitable for histogram plots.

If non-empty, we place dimension values into buckets after string to int64. Dimension values that are not the string representation of an integral value will be converted to zero. The bucket values have to be in increasing order. Each bucket is closed on the lower end, and open on the upper end. The "first" bucket includes all values less than the first boundary, the "last" bucket includes all values up to infinity. Dimension values that fall in a bucket get transformed to a new dimension value. For example, if one gives a list of "0, 1, 3, 4, 7", then we return the following buckets:

- bucket #1: values < 0, dimension value "<0"
- bucket #2: values in [0,1), dimension value "0"
- bucket #3: values in [1,3), dimension value "1-2"
- bucket #4: values in [3,4), dimension value "3"
- bucket #5: values in [4,7), dimension value "4-6"
- bucket #6: values >= 7, dimension value "7+

Examples

```r
# Not run:
library(googleAnalyticsR)

# authenticate, or use the RStudio Addin "Google API Auth" with analytics scopes set
ga_auth()

# get your accounts
account_list <- ga_account_list()

# account_list will have a column called "viewId"
account_list$viewId
```
## View account_list and pick the viewId you want to extract data from

ga_id <- 123456

# examine the meta table to see metrics and dimensions you can query
meta

## simple query to test connection
google_analytics(ga_id,
    date_range = c("2017-01-01", "2017-03-01"),
    metrics = "sessions",
    dimensions = "date"
)

## change the quotaUser to fetch under
options("googleAnalyticsR.quotaUser" = "test_user")
google_analytics(1234567, date_range = c("30daysAgo", "yesterday"), metrics = "sessions")

## End(Not run)

google_analytics_3

---

Get Google Analytics v3 data (formerly google_analytics())

---

### Description

Legacy v3 API, for more modern API use google_analytics.

### Usage

```r
google_analytics_3(
  id,
  start,
  end,
  metrics = c("sessions", "bounceRate"),
  dimensions = NULL,
  sort = NULL,
  filters = NULL,
  segment = NULL,
  samplingLevel = c("DEFAULT", "FASTER", "HIGHER_PRECISION"),
  max_results = 100,
  type = c("ga", "mcf")
)
```

### Arguments

- **id**: A character vector of View Ids to fetch from.
- **start**: Start date in YYY-MM-DD format.
end: End date in YYY-MM-DD format.
metrics: A character vector of metrics. With or without ga: prefix.
dimensions: A character vector of dimensions. With or without ga: prefix.
sort: How to sort the results, in form 'ga:sessions,-ga:bounceRate'
filters: Filters for the result, in form 'ga:sessions>0;ga:pagePath=~blah'
segment: How to segment.
samplingLevel: Choose "WALK" to mitigate against sampling.
max_results: Default 100. If greater than 10,000 then will batch GA calls.
type: ga = Google Analytics v3; mcf = Multi-Channel Funels.

Value

For one id a data.frame of data, with meta-data in attributes.

See Also

https://developers.google.com/analytics/devguides/reporting/core/v3/

Examples

## Not run:

library(googleAnalyticsR)

## Authenticate in Google OAuth2
## this also sets options
ga_auth()

## if you need to re-authenticate use ga_auth(new_user=TRUE)
## if you have your own Google Dev console project keys,
## then don't run ga_auth() as that will set to the defaults.
## instead put your options here, and run googleAuthR::gar_auth()

## get account info, including View Ids
account_list <- ga_account_list()
ga_id <- account_list$viewId[1]

## get a list of what metrics and dimensions you can use
meta <- ga_meta()
head(meta)

## pick the account_list$viewId you want to see data for.
## metrics and dimensions can have or have not "ga:" prefix

gadata <- google_analytics_3(id = ga_id,
               start="2015-08-01", end="2015-08-02",
               metrics = c("sessions", "bounceRate"),
## google_analytics_bq

Get Google Analytics 360 BigQuery data

### Description

Turn a google_analytics style call into BigQuery SQL. Used with Google Analytics 360 BigQuery exports.

### Usage

```r
google_analytics_bq( 
  projectId,
```


datasetId,
start = NULL,
end = NULL,
metrics = NULL,
dimensions = NULL,
sort = NULL,
filters = NULL,
max_results = 100,
query = NULL,
return_query_only = FALSE,
bucket = NULL,
download_file = NULL

Arguments

projectId The Google project Id where the BigQuery exports sit
datasetId DatasetId of GA export. This should match the GA View ID
start start date
end end date
metrics metrics to query
dimensions dimensions to query
sort metric to sort by
filters filter results
max_results How many results to fetch
query If query is non-NULL then it will use that and ignore above
return_query_only Only return the constructed query, don’t call BigQuery
bucket if over 100000 results, specify a Google Cloud bucket to send data to
download_file Where to save async files. If NULL saves to current working directory.

Details

All data will be unsampled, and requests will cost money against your BigQuery quota.

Requires installation of bigQueryR and authentication under ga_bq_auth() or googleAuthR::gar_auth() with BigQuery scope set. View your projectIds upon authentication via bigQueryR::bqr_list_projects()

No segments for now.

Goals are not specified in BQ exports, so you need to look at how you define them and replicate per view e.g. unique pageviews or unique events.

Custom dimensions can be specified as session or hit level, so ignoring the setting in GA interface.

You can get a sample Google Analytics dataset in bigquery by following the instructions here: https://support.google.com/analytics/answer/3416091?hl=en
make_cohort_group

Value
data.frame of results

See Also

https://support.google.com/analytics/answer/4419694?hl=en https://support.google.com/analytics/answer/3437719?hl=en

make_cohort_group  Create a cohort group

Description
Create a cohort group

Usage
make_cohort_group(cohorts, lifetimeValue = FALSE, cohort_types = NULL)

Arguments
cohorts A named list of start/end date pairs
lifetimeValue lifetimeValue TRUE or FALSE. Only works for webapps.
cohort_types placeholder, does nothing as only FIRST_VISIT_DATE supported.

Details
Example: list("cohort 1" = c("2015-08-01","2015-08-01"),"cohort 2" = c("2015-07-01","2015-07-01"))

Value
A cohortGroup object

See Also

https://developers.google.com/analytics/devguides/reporting/core/v4/advanced#cohort_and_lifetime_value_ltv_dimensions_and_metrics

Examples

## Not run:
library(googleAnalyticsR)

## authenticate,
## or use the RStudio Addin "Google API Auth" with analytics scopes set
ga_auth()
## get your accounts
account_list <- google_analytics_account_list()

## pick a profile with data to query
ga_id <- account_list[23,'viewId']

## first make a cohort group
cohort4 <- make_cohort_group(list("cohort 1" = c("2015-08-01", "2015-08-01"),
                                "cohort 2" = c("2015-07-01","2015-07-01")))

## then call cohort report. No date_range and must include metrics and dimensions
## from the cohort list
cohort_example <- google_analytics(ga_id,
                                 dimensions=c('cohort'),
                                 cohort = cohort4,
                                 metrics = c('cohortTotalUsers'))

### Lifetime Value report - just a variation of the cohort report
# with lifetimeValue = TRUE
### and ltv specific metrics
### The view MUST be an app view at the moment

## make a cohort group with lifetimeValue = TRUE
cohort_ltv <- make_cohort_group(list("cohort 1" = c("2018-12-01", "2018-12-31"),
                                    "cohort 2" = c("2019-01-01", "2019-01-31")),
                                lifetimeValue = TRUE)

## call a cohort report with ltv metrics
ltv_example <- google_analytics(ga_id,
                                dimensions = c('cohort', "acquisitionTrafficChannel"),
                                cohorts = cohort_ltv,
                                metrics = c("cohortGoalCompletionsPerUserWithLifetimeCriteria"))

## End(Not run)
Usage

meta

Format

A data frame containing metric and dimensions that you can query the Reporting API with.

Details

Running your own call will be more up to date, but this is here in case.

It does not include the multi-channel or cohort variables.

Source


---

meta4

Google Analytics API metadata

Description

This is a local copy of the data provided by ga_meta("data")

Usage

meta4

Format

A data frame containing metric and dimensions that you can query the Data API with.

Details

Running your own call will be more up to date, but this is here in case.

Source

https://developers.google.com/analytics/devguides/reporting/data/v1/api-schema
metricDimensionSelectUI

metricDimensionSelectUI - GA4 Shiny Module

Description

Create a Google Analytics variable selector
Shiny Module for use with GA4 metric and dimension fields fetched via ga_meta("ga4")

Usage

metricDimensionSelectUI(id, label = "Metric", multiple = TRUE, width = NULL)

metricDimensionSelect(
  id,
  field_type = c("metric", "dimension"),
  custom_meta = NULL,
  default = NULL
)

Arguments

id The Shiny id
label label
multiple multiple select
width width of select
field_type metric or dimension
custom_meta Pass a meta field table from ga_meta("ga4") to get custom fields from GA4 (reactive)
default The default selected choice. First element if NULL

Value

Shiny UI
the selected variable

See Also

Other Shiny modules: accountPickerUI(), authDropdownUI(), authDropdown(), multi_selectUI(), multi_select()
Examples

## Not run:

```r
# ui.R
metricDimensionSelect("mets1")
metricDimensionSelect("dims1")

#server.R
metrics <- metricDimensionSelect("mets1", "metric")
dims <- metricDimensionSelect("dims1", "dimension")

# use in app with custom fields
ui <- fluidPage(title = "Shiny App",
                accountPickerUI("auth_menu", inColumns = TRUE),
                metricDimensionSelectUI("mets1"),
                metricDimensionSelectUI("dims_custom")
)
server <- function(input, output, session){
  token <- gar_shiny_auth(session)
  accs <- reactive({
    req(token)
    ga_account_list("ga4")
  })
  # no custom data
  metrics <- metricDimensionSelect("mets1")
  # module for authentication
  property_id <- accountPicker("auth_menu", ga_table = accs, id_only = TRUE)
  meta <- reactive({
    req(property_id())
    ga_meta("data", propertyId = property_id())
  })
  # custom data
  dims_custom <- metricDimensionSelect("dims_custom",
                                      type = "dimension",
                                      custom_meta = meta())
  shinyApp(gar_shiny_ui(ui, login_ui = silent_auth), server)

## End(Not run)
Description

Make a metric filter object

Usage

```r
met_filter(
  metric,
  operator = c("EQUAL", "LESS_THAN", "GREATER_THAN", "IS_MISSING"),
  comparisonValue,
  not = FALSE
)
```

Arguments

- `metric` metric name to filter on.
- `operator` How to match the dimension.
- `comparisonValue` What to match.
- `not` Logical NOT operator. Boolean.

Value

An object of class `met_fil_ga4` for use in `filter_clause_ga4()`

See Also

Other filter functions: `dim_filter()`, `filter_clause_ga4()`

Examples

```r
## Not run:
library(googleAnalyticsR)

## authenticate,
## or use the RStudio Addin "Google API Auth" with analytics scopes set
ga_auth()

## get your accounts
account_list <- google_analytics_account_list()

## pick a profile with data to query
ga_id <- account_list[23,'viewId']
```
## create filters on metrics

mf <- met_filter("bounces", "GREATER_THAN", 0)
mf2 <- met_filter("sessions", "GREATER", 2)

## create filters on dimensions

df <- dim_filter("source","BEGINS_WITH","1", not = TRUE)
df2 <- dim_filter("source","BEGINS_WITH","a", not = TRUE)

## construct filter objects

fc2 <- filter_clause_ga4(list(df, df2), operator = "AND")
fci <- filter_clause_ga4(list(mf, mf2), operator = "AND")

## make v4 request

ga_data1 <- google_analytics_4(ga_id, 
  date_range = c("2015-07-30","2015-10-01"), 
  dimensions=c('source','medium'), 
  metrics = c('sessions','bounces'), 
  met_filters = fc, 
  dim_filters = fc2, 
  filtersExpression = "ga:source!=(direct)"
)

## End(Not run)

---

**multi_select**

**multi_select Shiny Module**

**Description**

Shiny Module for use with multi_selectUI

**Usage**

```r
multi_select( 
  input, 
  output, 
  session, 
  type = c("METRIC", "DIMENSION"), 
  subType = c("all", "segment", "cohort"), 
  default = NULL 
)
```

**Arguments**

- `input`: shiny input
- `output`: shiny output
- `session`: shiny session
**multi_selectUI**

multi_select UI Shiny Module

**Description**

Shiny Module for use with `multi_select`

**Usage**

```r
multi_selectUI(id, label = "Metric", multiple = TRUE, width = NULL)
```

**Arguments**

- `id`  
  Shiny id
- `label`  
  label
- `multiple`  
  multiple select
- `width`  
  width of select

**Details**

Create a Google Analytics variable selector

**Value**

Shiny UI

**See Also**

Other Shiny modules: `accountPickerUI()`, `authDropdownUI()`, `authDropdown()`, `metricDimensionSelectUI()`, `multi_select()`
order_type

Make an OrderType object

Description
Make an OrderType object

Usage
order_type(
  field,
  sort_order = c("ASCENDING", "DESCENDING"),
  orderType = c("VALUE", "DELTA", "SMART", "HISTOGRAM_BUCKET", "DIMENSION_AS_INTEGER")
)

Arguments
- field: One field to sort by
- sort_order: ASCENDING or DESCENDING
- orderType: Type of ordering

Details
For multiple order sorting, create separate OrderType objects to pass

Value
A order_type_ga4 object for use in GAv4 fetch

pivot_ga4

Make a pivot object

Description
Make a pivot object

Usage
pivot_ga4(
  pivot_dim,
  metrics,
  dim_filter_clause = NULL,
  startGroup = 0,
  maxGroupCount = 5
)
Arguments

pivot_dim  A character vector of dimensions
metrics    Metrics to aggregate and return.
dim_filter_clause Only data included in filter included.
startGroup which groups of k columns are included in response (0 indexed).
maxGroupCount Maximum number of groups to return.

Details

If maxGroupCount is set to -1 returns all groups.

Value

pivot object of class pivot_ga4 for use in filter_clause_ga4()

Examples

## Not run:
library(googleAnalyticsR)

## authenticate,
## or use the RStudio Addin "Google API Auth" with analytics scopes set
ga_auth()

## get your accounts
account_list <- google_analytics_account_list()

## pick a profile with data to query
ga_id <- account_list[23,'viewId']

## filter pivot results to
pivot_dim_filter1 <- dim_filter("medium",
   "REGEXP",
   "organic|social|email|cpc")

pivot_dim_clause <- filter_clause_ga4(list(pivot_dim_filter1))
pivme <- pivot_ga4("medium",
   metrics = c("sessions"),
   maxGroupCount = 4,
   dim_filter_clause = pivot_dim_clause)

pivtest <- google_analytics(ga_id,
   c("2016-01-30","2016-10-01"),
   dimensions=c('source'),
segmentBuilder

```r
metrics = c('sessions'),
pivots = list(pivme))
```

## End(Not run)

---

**segmentBuilder**  
*Create a GAv4 Segment Builder*

---

**Description**

Shiny Module for use with `segmentBuilderUI`

**Usage**

```r
segmentBuilder(input, output, session)
```

**Arguments**

- **input**: shiny input
- **output**: shiny output
- **session**: shiny session

**Details**

Call via `shiny::callModule(segmentBuilder,"your_id")`

**Value**

A segment definition

**Examples**

```r
## Not run:
library(shiny)
library(googleAnalyticsR)

ui <- shinyUI(fluidPage(
  segmentBuilderUI("test1")
))

server <- shinyServer(function(input, output, session) {
  segment <- callModule(segmentBuilder, "test1")
})
```
.. use segment() in further gav4 calls.
}

# Run the application
shinyApp(ui = ui, server = server)

## End(Not run)

---

**segmentBuilderUI**

*Create a GAv4 Segment Builder*

**Description**

Shiny Module for use with `segmentBuilder`

**Usage**

```r
segmentBuilderUI(id)
```

**Arguments**

- `id` (Shiny id)

**Value**

Shiny UI for use in app

**Examples**

```r
## Not run:
library(shiny)
library(googleAnalyticsR)

ui <- shinyUI(fluidPage(
  segmentBuilderUI("test1")
))

server <- shinyServer(function(input, output, session) {
  segment <- callModule(segmentBuilder, "test1")
  .. use segment() in further gav4 calls.
```

Make a segment definition

Description

Defines the segment to be a set of SegmentFilters which are combined together with a logical AND operation.

segment_define is in the hierarchy of segment creation, for which you will also need:

- segment_define: AND combination of SegmentFilters
- segment_vector_simple or segment_vector_sequence
- segment_element that are combined in OR lists for segment_vectors_*

Usage

segment_define(segment_filters, not_vector = NULL)

Arguments

segment_filters
  A list of segment_vector_simple and segment_vector_sequence

not_vector
  Boolean applied to each SegmentFilter step. If NULL, assumed FALSE

Value

segmentDefinition object for segment_ga4

See Also

Other v4 segment functions: segment_element(), segment_ga4, segment_vector_sequence(), segment_vector_simple()
segment_element

Description

segment_element is the lowest hierarchy of segment creation, for which you will also need:

- segment_define : AND combination of segmentFilters
- segment_vector_simple or segment_vector_sequence
- segment_element that are combined in OR lists for segment_vectors_*

Usage

```r
segment_element(
  name,
  type = c("METRIC", "DIMENSION"),
  not = FALSE,
  expressions = NULL,
  caseSensitive = NULL,
  minComparisonValue = NULL,
  maxComparisonValue = NULL,
  scope = c("SESSION", "USER", "HIT", "PRODUCT"),
  comparisonValue = NULL,
  matchType = c("PRECEDES", "IMMEDIATELY_PRECEDES")
)
```

Arguments

- **name**: Name of the GA metric or dimension to segment on
- **operator**: How name shall operate on expression or comparisonValue
- **type**: A metric or dimension based segment element
- **not**: Should the element be the negation of what is defined
- **expressions**: What the name shall compare to
- **caseSensitive**: Whether to be case sensitive
- **minComparisonValue**: Minimum comparison values for BETWEEN
- **maxComparisonValue**: Max comparison value for BETWEEN operator
- **scope**: Scope of the metric value
- **comparisonValue**: What the name shall compare to
- **matchType**: If used in sequence segment, what behaviour
segment_ga4

Value

A SegmentFilterClause object

See Also

Other v4 segment functions: segment_define(), segment_ga4, segment_vector_sequence(), segment_vector_simple()

segment_ga4 Make a segment object for use

Description

A Segment is a subset of the Analytics data. For example, of the entire set of users, one Segment might be users from a particular country or city.

Usage

segment_ga4(
    name,
    segment_id = NULL,
    user_segment = NULL,
    session_segment = NULL
)

Arguments

name The name of the segment for the reports.
segment_id The segment ID of a built in or custom segment e.g. ga:id:-3
user_segment A list of segment_define’s that apply to users
session_segment A list of segment_define’s that apply to sessions

Details

segment_ga4 is the top hierarchy of segment creation, for which you will also need:

- segment_define : AND combination of segmentFilters
- segment_vector_simple or segment_vector_sequence
- segment_element that are combined in OR lists for segment_vectors_*

Value

A segmentFilter object. You can pass a list of these to the request.
See Also

Other v4 segment functions: `segment_define()`, `segment_element()`, `segment_vector_sequence()`, `segment_vector_simple()`

Examples

```r
## Not run:
library(googleAnalyticsR)

## authenticate,
## or use the RStudio Addin "Google API Auth" with analytics scopes set
ga_auth()

## get your accounts
account_list <- google_analytics_account_list()

## pick a profile with data to query
ga_id <- account_list[23,"ViewId"]

## make a segment element
se <- segment_element("sessions",
  operator = "GREATER_THAN",
  type = "METRIC",
  comparisonValue = 1,
  scope = "USER")

se2 <- segment_element("medium",
  operator = "EXACT",
  type = "DIMENSION",
  expressions = "organic")

## choose between segment_vector_simple or segment_vector_sequence
## Elements can be combined into clauses, which can then be
## combined into OR filter clauses
sv_simple <- segment_vector_simple(list(list(se)))
sv_simple2 <- segment_vector_simple(list(list(se2)))

## Each segment vector can then be combined into a logical AND
seg_defined <- segment_define(list(sv_simple, sv_simple2))

## if only one AND definition, you can leave out wrapper list()
seg_defined_one <- segment_define(sv_simple)

## Each segment definition can apply to users, sessions or both.
```
## You can pass a list of several segments

```
segment4 <- segment_ga4("simple", user_segment = seg_defined)
```

## Add the segments to the segments param

```
segment_example <- google_analytics(ga_id,
    c("2015-07-30","2015-10-01"),
    dimensions=c('source','medium','segment'),
    segments = segment4,
    metrics = c('sessions','bounces')
)
```

## Sequence segment

```
se2 <- segment_element("medium",
    operator = "EXACT",
    type = "DIMENSION",
    expressions = "organic")

se3 <- segment_element("medium",
    operator = "EXACT",
    type = "DIMENSION",
    not = TRUE,
    expressions = "organic")
```

## step sequence

## users who arrived via organic then via referral
```
sv_sequence <- segment_vector_sequence(list(list(se2),
    list(se3)))
```

```
seq_defined2 <- segment_define(list(sv_sequence))
```

```
segment4_seq <- segment_ga4("sequence", user_segment = seq_defined2)
```

## Add the segments to the segments param

```
segment_seq_example <- google_analytics(ga_id,
    c("2016-04-01","2016-05-01"),
    dimensions=c('source','segment'),
    segments = segment4_seq,
    metrics = c('sessions','bounces')
)
```

## End(Not run)
segment_vector_simple

Description

segment_vector_sequence is in the hierarchy of segment creation, for which you will also need:

- `segment_define`: AND combination of segmentFilters
- `segment_vector_simple` or `segment_vector_sequence`
- `segment_element` that are combined in OR lists for `segment_vectors_*`

Usage

```r
segment_vector_sequence(segment_elements, firstStepMatch = FALSE)
```

Arguments

- `segment_elements`: a list of OR lists of `segment_element`
- `firstStepMatch`: `FALSE` default

See Also

Other v4 segment functions: `segment_define()`, `segment_element()`, `segment_ga4`, `segment_vector_simple()`

---

segment_vector_simple  Make a simple segment vector

Description

`segment_vector_simple` is in the hierarchy of segment creation, for which you will also need:

- `segment_define`: AND combination of segmentFilters
- `segment_vector_simple` or `segment_vector_sequence`
- `segment_element` that are combined in OR lists for `segment_vectors_*`

Usage

```r
segment_vector_simple(segment_elements)
```

Arguments

- `segment_elements`: A list of OR lists of `segment_element`

Value

A segment vector you can put in a list for use in `segment_ga4`

See Also

Other v4 segment functions: `segment_define()`, `segment_element()`, `segment_ga4`, `segment_vector_sequence()`
Index

* GA modelling functions
  ga_model, 58
  ga_model_edit, 59
  ga_model_example, 60
  ga_model_load, 61
  ga_model_make, 61
  ga_model_save, 64
  ga_model_shiny, 65
  ga_model_shiny_load, 68
  ga_model_shiny_template, 69
  ga_model_write, 70

* GA4 functions
  ga_data, 34
  ga_data_filter, 37
  ga_data_order, 40

* GAv4 fetch functions
  google_analytics, 93

* Google Ad management functions
  ga_adwords, 12
  ga_adwords_add_linkid, 12
  ga_adwords_delete_linkid, 13
  ga_adwords_list, 14

* Measurement Protocol functions
  ga_mp_cid, 70
  ga_mp_event, 71
  ga_mp_event_item, 71
  ga_mp_send, 73

* Shiny modules
  accountPickerUI, 4
  authDropdown, 5
  authDropdownUI, 6
  metricDimensionSelectUI, 104
  multi_select, 107
  multi_selectUI, 108

* User management functions
  ga_users_add, 84
  ga_users_delete, 85
  ga_users_delete_linkid, 86
  ga_users_list, 88
  ga_users_update, 89

* account structure functions
  ga_account_list, 11
  ga_accounts, 10
  ga_view, 90
  ga_view_list, 91
  ga_webproperty, 92
  ga_webproperty_list, 92

* clientid functions
  ga_clientid_activity, 19
  ga_clientid_activity_unnest, 21
  ga_clientid_deletion, 22
  ga_clientid_hash, 24

* custom datasource functions
  ga_custom_datasource, 24
  ga_custom_upload, 25
  ga_custom_upload_delete, 26
  ga_custom_upload_file, 27
  ga_custom_upload_list, 29

* custom variable functions
  ga_custom_vars, 30
  ga_custom_vars_create, 30
  ga_custom_vars_list, 32
  ga_custom_vars_patch, 33

* datasets
  meta, 102
  meta4, 103

* filter functions
  dim_filter, 7
  filter_clause_ga4, 8
  met_filter, 106

* filter management functions
  ga_filter, 43
  ga_filter_delete, 46
  ga_filter_list, 47
  ga_filter_view, 50
  ga_filter_view_list, 51

* goal management functions
  ga_goal, 52
INDEX

121

ga_model_make, 58, 60, 61, 64, 67, 69, 70
ga_model_refresh, 63
ga_model_save, 58, 60–62, 64, 67, 69, 70

multi_select, 5–7, 104, 107, 108
multi_selectUI, 5–7, 104, 107, 108

order_type, 94, 109

pivot_ga4, 94, 109

segment_define, 113, 113, 114–116, 118
segment_element, 113, 114, 114, 115, 116, 118

segment_ga4, 94, 113, 115, 115, 118
segment_vector_sequence, 113–116, 117, 118

segment_vector_simple, 113–116, 118, 118
segmentBuilder, 111, 112
segmentBuilderUI, 111, 112
Startup, 17
Sys.setenv, 17

whisker.render, 66

meta, 103

metricDimensionSelect

metricDimensionSelectUI, 104

meta4, 103

metricDimensionSelectUI, 5–7, 104, 108

met_filter, 7–9, 94, 106

meta, 94, 102