Package ‘googleAnalyticsR’

November 4, 2019

Type Package
Version 0.7.1
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 and Multi-Channel Funnel API.
URL http://code.markedmondson.me/googleAnalyticsR/
BugReports https://github.com/MarkEdmondson1234/googleAnalyticsR/issues
Depends R (>= 3.3.0)
Imports assertthat (>= 0.2.0), dplyr (>= 0.8.0), googleAuthR (>= 1.1.1), httr (>= 1.3.1), jsonlite (>= 1.5), magrittr (>= 1.5), memoise, methods, purrr (>= 0.2.2), rlang (>= 0.1.0), stats, tibble (>= 2.0.1), tidyverse (>= 1.0.0), utils
Suggests bigQueryR (>= 0.3.1), covr, formatR, ganalytics, googleCloudStorageR (>= 0.2.0), htmlwidgets, httptest (>= 3.2.2), knitr, miniUI (>= 0.1.1), rmarkdown, shiny (>= 0.13.2)
License MIT + file LICENSE
LazyData TRUE
RoxygenNote 6.1.1
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]
Maintainer Mark Edmondson <m@sunholo.com>
Repository CRAN
Date/Publication 2019-11-04 10:10:02 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>authDropdown</td>
<td>4</td>
</tr>
<tr>
<td>authDropdownUI</td>
<td>5</td>
</tr>
<tr>
<td>dim_filter</td>
<td>5</td>
</tr>
<tr>
<td>fetch_google_analytics_4</td>
<td>7</td>
</tr>
<tr>
<td>fetch_google_analytics_4_slow</td>
<td>8</td>
</tr>
<tr>
<td>filter_clause_ga4</td>
<td>9</td>
</tr>
<tr>
<td>ga_accounts</td>
<td>10</td>
</tr>
<tr>
<td>ga_account_list</td>
<td>11</td>
</tr>
<tr>
<td>ga_adwords</td>
<td>12</td>
</tr>
<tr>
<td>ga_adwords_add_linkid</td>
<td>12</td>
</tr>
<tr>
<td>ga_adwords_delete_linkid</td>
<td>13</td>
</tr>
<tr>
<td>ga_adwords_list</td>
<td>14</td>
</tr>
<tr>
<td>ga_aggregate</td>
<td>15</td>
</tr>
<tr>
<td>ga_allowed_metric_dim</td>
<td>16</td>
</tr>
<tr>
<td>ga_auth</td>
<td>16</td>
</tr>
<tr>
<td>ga_cache_call</td>
<td>18</td>
</tr>
<tr>
<td>ga_clientid_activity</td>
<td>19</td>
</tr>
<tr>
<td>ga_clientid_activity_unnest</td>
<td>20</td>
</tr>
<tr>
<td>ga_clientid_deletion</td>
<td>22</td>
</tr>
<tr>
<td>ga_clientid_hash</td>
<td>23</td>
</tr>
<tr>
<td>ga_custom_datasource</td>
<td>24</td>
</tr>
<tr>
<td>ga_custom_upload</td>
<td>25</td>
</tr>
<tr>
<td>ga_custom_upload_delete</td>
<td>26</td>
</tr>
<tr>
<td>ga_custom_upload_file</td>
<td>27</td>
</tr>
<tr>
<td>ga_custom_upload_list</td>
<td>28</td>
</tr>
<tr>
<td>ga_custom_vars</td>
<td>29</td>
</tr>
<tr>
<td>ga_custom_vars_create</td>
<td>30</td>
</tr>
<tr>
<td>ga_custom_vars_list</td>
<td>31</td>
</tr>
<tr>
<td>ga_custom_vars_patch</td>
<td>32</td>
</tr>
<tr>
<td>ga_experiment</td>
<td>33</td>
</tr>
<tr>
<td>ga_experiment_list</td>
<td>34</td>
</tr>
<tr>
<td>ga_filter</td>
<td>34</td>
</tr>
<tr>
<td>ga_filter_add</td>
<td>35</td>
</tr>
<tr>
<td>ga_filter_apply_to_view</td>
<td>37</td>
</tr>
<tr>
<td>ga_filter_delete</td>
<td>38</td>
</tr>
<tr>
<td>ga_filter_list</td>
<td>38</td>
</tr>
<tr>
<td>ga_filter_update</td>
<td>39</td>
</tr>
<tr>
<td>ga_filter_update_filter_link</td>
<td>40</td>
</tr>
<tr>
<td>ga_filter_view</td>
<td>41</td>
</tr>
<tr>
<td>ga_filter_view_list</td>
<td>42</td>
</tr>
<tr>
<td>ga_goal</td>
<td>43</td>
</tr>
<tr>
<td>ga_goal_add</td>
<td>43</td>
</tr>
<tr>
<td>ga_goal_list</td>
<td>46</td>
</tr>
<tr>
<td>ga_goal_update</td>
<td>46</td>
</tr>
<tr>
<td>ga_meta</td>
<td>48</td>
</tr>
<tr>
<td>ga_model</td>
<td>48</td>
</tr>
<tr>
<td>R topics documented:</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>ga_model_edit</td>
<td>49</td>
</tr>
<tr>
<td>ga_model_example</td>
<td>49</td>
</tr>
<tr>
<td>ga_model_load</td>
<td>50</td>
</tr>
<tr>
<td>ga_model_make</td>
<td>50</td>
</tr>
<tr>
<td>ga_model_save</td>
<td>52</td>
</tr>
<tr>
<td>ga_model_tweet</td>
<td>53</td>
</tr>
<tr>
<td>ga_model_write</td>
<td>54</td>
</tr>
<tr>
<td>ga_remarketing_build</td>
<td>55</td>
</tr>
<tr>
<td>ga_remarketing_create</td>
<td>56</td>
</tr>
<tr>
<td>ga_remarketing_estimate</td>
<td>57</td>
</tr>
<tr>
<td>ga_remarketing_get</td>
<td>58</td>
</tr>
<tr>
<td>ga_remarketing_list</td>
<td>59</td>
</tr>
<tr>
<td>ga_segment_list</td>
<td>59</td>
</tr>
<tr>
<td>ga_unsampled</td>
<td>60</td>
</tr>
<tr>
<td>ga_unsampled_download</td>
<td>60</td>
</tr>
<tr>
<td>ga_unsampled_list</td>
<td>62</td>
</tr>
<tr>
<td>ga_users_add</td>
<td>63</td>
</tr>
<tr>
<td>ga_users_delete</td>
<td>64</td>
</tr>
<tr>
<td>ga_users_delete_linkid</td>
<td>65</td>
</tr>
<tr>
<td>ga_users_list</td>
<td>66</td>
</tr>
<tr>
<td>ga_users_update</td>
<td>67</td>
</tr>
<tr>
<td>ga_view</td>
<td>68</td>
</tr>
<tr>
<td>ga_view_list</td>
<td>69</td>
</tr>
<tr>
<td>ga_webproperty</td>
<td>70</td>
</tr>
<tr>
<td>googleAnalyticsR</td>
<td>71</td>
</tr>
<tr>
<td>google_analytics</td>
<td>72</td>
</tr>
<tr>
<td>google_analytics_3</td>
<td>74</td>
</tr>
<tr>
<td>google_analytics_bq</td>
<td>77</td>
</tr>
<tr>
<td>make_cohort_group</td>
<td>78</td>
</tr>
<tr>
<td>make_ga_4_req</td>
<td>80</td>
</tr>
<tr>
<td>meta</td>
<td>82</td>
</tr>
<tr>
<td>met_filter</td>
<td>83</td>
</tr>
<tr>
<td>multi_select</td>
<td>84</td>
</tr>
<tr>
<td>multi_selectUI</td>
<td>85</td>
</tr>
<tr>
<td>order_type</td>
<td>86</td>
</tr>
<tr>
<td>pivot_ga4</td>
<td>86</td>
</tr>
<tr>
<td>segmentBuilder</td>
<td>88</td>
</tr>
<tr>
<td>segmentBuilderUI</td>
<td>89</td>
</tr>
<tr>
<td>segment_define</td>
<td>90</td>
</tr>
<tr>
<td>segment_element</td>
<td>90</td>
</tr>
<tr>
<td>segment_ga4</td>
<td>91</td>
</tr>
<tr>
<td>segment_vector_sequence</td>
<td>94</td>
</tr>
<tr>
<td>segment_vector_simple</td>
<td>95</td>
</tr>
</tbody>
</table>

Index 96
authDropdown

authDropdown [Shiny Module]

Description

Shiny Module for use with authDropdownUI

Usage

authDropdown(input, output, session, ga.table, viewIdOnly = TRUE, rmNA = TRUE)

Arguments

input        shiny input
output       shiny output
session      shiny session
ga.table     A table of GA tables
viewIdOnly   Default only returns the viewId, set to FALSE to return the row of ga.table satisfying the selections
rmNA         Will remove any rows that have NA listed for the columns. Set to FALSE to return all rows.

Details

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

Value

GA View Id selected

See Also

Other Shiny modules: authDropdownUI, multi_selectUI, multi_select
authDropdownUI

authDropdownUI [Shiny Module]

Description

Makes a dropdown row for use for authentication.

Usage

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

Arguments

id
Shiny id.
width
The width of the input
inColumns
whether to wrap selectInputs in width=4 columns.
Shiny Module for use with authDropdown.

Value

Shiny UI

See Also

Other Shiny modules: authDropdown, multi_selectUI, multi_select

dim_filter

Make a dimension filter object

Description

Make a dimension filter object

Usage

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_filter_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,
derange = 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)
```
fetch_google_analytics_4

Fetch multiple GAv4 requests

Description

Fetch the GAv4 requests as created by make_ga_4_req

Usage

fetch_google_analytics_4(request_list, merge = FALSE, useResourceQuotas = NULL)

Arguments

request_list  A list of requests created by make_ga_4_req
merge          If TRUE then will rbind that list of data.frames
useResourceQuotas
              If using GA360, access increased sampling limits. Default NULL, set to TRUE or FALSE if you have access to this feature.

Details

For same viewId, daterange, segments, samplingLevel and cohortGroup, v4 batches can be made

Value

A dataframe if one request, or a list of data.frames if multiple.

See Also

Other GAv4 fetch functions: fetch_google_analytics_4_slow, google_analytics, make_ga_4_req

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 <- ga_account_list()

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

ga_req1 <- make_ga_4_req(ga_id,
  date_range = c("2015-07-30","2015-10-01"),
  dimensions=c('source','medium'),
  metrics = c('sessions'))

ga_req2 <- make_ga_4_req(ga_id,
  date_range = c("2015-07-30","2015-10-01"),
  dimensions=c('source','medium'),
  metrics = c('users'))

fetch_google_analytics_4(list(ga_req1, ga_req2))

## End(Not run)

---

**fetch_google_analytics_4_slow**

*Fetch GAv4 requests one at a time*

**Description**

Due to large complicated queries causing the v4 API to timeout, this option is added to fetch via the more traditional one report per request.

**Usage**

```r
fetch_google_analytics_4_slow(request_list, max_rows, allRows = FALSE, useResourceQuotas = NULL)
```

**Arguments**

- `request_list`: A list of requests created by `make_ga_4_req`
- `max_rows`: Number of rows requested (if not fetched)
- `allRows`: Whether to fetch all available rows
- `useResourceQuotas`: If using GA360, access increased sampling limits. Default NULL, set to TRUE or FALSE if you have access to this feature.

**Value**

A dataframe of all the requests

**See Also**

Other GAv4 fetch functions: `fetch_google_analytics_4`, `google_analytics`, `make_ga_4_req`
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, then pass the objects to make_ga_4_req

Value

An object of class dim_fil_ga4 or met_fil_ga4 for use in make_ga_4_req

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()

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")
fenes <- 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)

gga_accounts

### ga_accounts

List account metadata

---

**Description**

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

**Usage**

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()
```

**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

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

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

---

**ga_adwords**  
*Get AdWords Link meta data*

---

**Description**  
Get AdWords Link meta data

**Usage**  

```javascript
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 an Adwords (Google ads) account and a Google Analytics property so that Adwords data can be accessed via Google Analytics and vice versa.

**Usage**  

```javascript
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 and Adwords (Google ads) account and a Google Analytics property

### Usage

```r
ga_adwords_delete_linkid(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>webPropertyAdWordsLinkId</td>
</tr>
</tbody>
</table>

### 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

```r
## Not run:
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

`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() (default: `"^avg|^percent|Rate$|^CPC$|^CTR$|^CPM$|^RPC$|^ROI$|^ROAS$|Per"`) 

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 `aggregateGADAta` so you can on the fly create summary data
# use aggregateGADAta so you can on the fly create summary data
google_analytics(81416156, date_range = c("10daysAgo", "yesterday"),
metrics = "sessions", dimensions = c("hour","date"))

# if we want totals per hour over the dates:
google_analytics(81416156, date_range = c("10daysAgo", "yesterday"),
meters = "sessions", dimensions = c("hour","date"))

# if we want totals per hour over the dates:
google_analytics(81416156, date_range = c("10daysAgo", "yesterday"),
meters = "sessions", dimensions = c("hour","date"))

# it knows not to sum metrics that are rates:
google_analytics(81416156, date_range = c("10daysAgo", "yesterday"),
meters = "sessions", dimensions = c("hour","date"))
```

## 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
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
          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
ga_auth(token = NULL, email = 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.
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

By 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"
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

## End(Not run)

---

**ga_cache_call**  
*Setup caching of API calls*

**Description**

Lets you cache API calls to disk

**Usage**

`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.

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](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("1106980347.1461227730", "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*
**ga_clientid_activity_unnest**

**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**

- `hits`: The hits data.frame with the columns to expand
- `column`: Which column to expand - one of "customDimension", "ecommerce", "goals"

**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.
**ga_clientid_hash**

**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**

```r
## 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(new_user = TRUE)

# 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

## End(Not run)
```

---

**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
**ga_custom_datasource**

**Value**

hashedClientId object list

**See Also**

Other clientid functions: `ga_clientid_activity_unnest, ga_clientid_activity, ga_clientid_deletion`

---

**ga_custom_datasource**  
*List Custom Data Sources*

**Description**

Get a list of custom data sources you have configured in Google Analytics web UI.

**Usage**

`ga_custom_datasource(accountId, webPropertyId)`

**Arguments**

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

**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`
**ga_custom_upload**

**Custom Data Source Upload Status**

**Description**

Get the status of a custom upload

**Usage**

```r
ga_custom_upload(accountId, webPropertyId, customDataSourceId, uploadId, upload_object)
```

**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: `ga_custom_datasource,ga_custom_upload_delete,ga_custom_upload_file,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",
                     
```

```r
obj <- ga_custom_upload_file(47850439,  "UA-4748043-2",
                     ",_jDsJHSFSU-uw038Bh8fUg",
                     
```
## obj will initially have status = PENDING

```r
go_custom_upload(upload_me)
```

## Send obj to `ga_custom_upload()` to check and renew status

```r
obj <- ga_custom_upload(upload_object = obj)
```

## End(Not run)

---

### ga_custom_upload_delete

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

**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

<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>
<tr>
<td>customDataImportUids</td>
<td>vector of file upload ids.</td>
</tr>
</tbody>
</table>
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**

<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>
<tr>
<td>upload</td>
<td>An R data.frame or a file path location (character)</td>
</tr>
</tbody>
</table>

**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",
                              "_jDsJHSFU-uw038Bh8fUg",
                              upload_me)

## obj will initially have status = PENDING
obj

==Google Analytics Custom Data Source Upload==
Custom Data Source ID: _jDsJHSFU-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: _jDsJHSFU-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**

`ga_custom_upload_list(accountId, webPropertyId, customDataSourceId)`
Arguments

accountId    Account Id
webPropertyId  Web Property Id
customDataSourceId  Custom data source Id

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**  Get Custom Dimensions or Metrics

Description

Get Custom Dimensions or Metrics

Usage

```r
ga_custom_vars(accountId, webPropertyId, type = c("customMetrics", "customDimensions"), customId)
```

Arguments

accountId    Account Id
webPropertyId  Web Property Id
type    A customMetric or customDimension
customId    The customMetricId or customDimensionId

Value

Custom Metric or Dimension meta data

See Also

Other custom variable functions: ga_custom_vars_create, ga_custom_vars_list, ga_custom_vars_patch
ga_custom_vars_create  

Create a custom dimension

Description

Create a dimension by specifying its attributes.

Usage

    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

    ## 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")

    ## End(Not run)
ga_custom_vars_list  List Custom Dimensions or Metrics

Description
List Custom Dimensions or Metrics

Usage

\[
\text{ga\_custom\_vars\_list(accountId, webPropertyId, type = c("customDimensions", "customMetrics"))}
\]

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: \textit{ga\_custom\_vars\_create, ga\_custom\_vars\_patch, ga\_custom\_vars}

Examples

```r
## 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)
```
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")`
ga_experiment_list  List Experiments

Description
List Experiments

Usage

\[
ga\_experiment\_list(accountId, \text{webPropertyId}, profileId)
\]

Arguments
- \text{accountId}  Account Id
- \text{webPropertyId}  Web Property Id
- \text{profileId}  Profile Id

Value
Experiments List

See Also
Other managementAPI 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
- \text{accountId}  Account Id
- \text{filterId}  Filter Id

Value
filter list
**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 propetyId 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**

- `ga_filter_delete, ga_filter_list, ga_filter_view_list, ga_filter_view`
- [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

## 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'
  )
)

## End(Not run)

---

**ga_filter_apply_to_view**

Apply an existing filter to view.

### Description
Apply an existing filter to view.

### Usage

```r
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: `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
```r
ga_filter_delete(accountId, webPropertyId = NULL, viewId = NULL, filterId, removeFromView = FALSE)
```

Arguments
- `accountId`: Account Id of the account that contains the filter
- `webPropertyId`: Property Id of the property that contains the filter
- `viewId`: View Id of the view that contains the filter
- `filterId`: Filter Id of the filter to be deleted
- `removeFromView`: Default if FALSE. If TRUE, deletes the filter from the view

Value
 TRUE if successful

See Also
Other filter management functions: `ga_filter_list`, `ga_filter_view_list`, `ga_filter_view`, `ga_filter`

---

ga_filter_list  
List filters for account

Description
List filters for account

Usage
```r
ga_filter_list(accountId)
```

Arguments
- `accountId`: Account Id
Value

filter list

See Also

Other filter management functions: ga_filter_delete, ga_filter_view_list, ga_filter_view, ga_filter

---

ga_filter_update  Updates an existing filter.

Description

Updates an existing filter.

Usage

```r
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

`ga_filter_view(accountId, webPropertyId, viewId, linkId)`

Arguments

- `accountId`  Account Id
- `webPropertyId`  Web Property Id
- `viewId`  Profile Id
- `linkId`  Link Id

Value

filter list

See Also

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

---

`ga_filter_view_list`  *List filters for view (profile)*

Description

List filters for view (profile)

Usage

`ga_filter_view_list(accountId, webPropertyId, viewId)`

Arguments

- `accountId`  Account Id
- `webPropertyId`  Web Property Id
- `viewId`  Profile Id

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**

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

**Arguments**

- **accountId**
  - Account Id
- **webPropertyId**
  - Web Property Id
- **profileId**
  - Profile Id
- **goalId**
  - Goal Id

**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**

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

**Arguments**

- **Goal**
  - The Goal object to be added to the view. See examples.
- **accountId**
  - Account Id of the account to add the Goal to
- **webPropertyId**
  - Property Id of the property to add the Goal to
- **viewId**
  - View Id of the view to add the Goal to
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,
```
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

### Description
List goals

### Usage

```
ga_goal_list(accountId, webPropertyId, profileId)
```

### Arguments
- `accountId`: Account Id
- `webPropertyId`: Web Property Id
- `profileId`: Profile Id

### Value
Goal list

### See Also
Other goal management functions: `ga_goal_add`, `ga_goal_update`, `ga_goal`

---

ga_goal_update

### Description
Updates an existing goal.

### Usage

```
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:

# Change the goal 11 to visits over 3 minutes
Goal <- list(
  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)

# Change destination url for goal 17
Goal <- list(
  urlDestinationDetails = list(
    url = '/checkout/success'
  )
)
ga_goal_update(Goal, accountId, propertyId, viewId, 17, method = "PATCH")

## End(Not run)
ga_meta

*Get current dimensions and metrics available in GA API.*

**Description**

Get current dimensions and metrics available in GA API.

**Usage**

```
ga_meta()
```

**Value**

dataframe of dimensions and metrics available to use

**See Also**

https://developers.google.com/analytics/devguides/reporting/metadata/v3/reference/metadata/columns/list

---

ga_model

*Use a model function created by ga_model_make*

**Description**

Use a model function created by ga_model_make

**Usage**

```
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_tweet`, `ga_model_write`
ga_model_edit  Edit a created ga_model

Description

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

Usage

```
ga_model_edit(model, data_f = NULL, required_columns = NULL,
              model_f = NULL, required_packages = NULL, description = NULL,
              outputShiny = NULL, renderShiny = 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**: A shiny render function that will create the output for outputShiny from output_f
- **output_f**: A function that inputs the output from model_f, outputs a visualisation

See Also

Other GA modelling functions: `ga_model_example, ga_model_load, ga_model_make, ga_model_save, ga_model_tweet, ga_model_write, ga_model`

---

ga_model_example  Load an example model

Description

Load an example model

Usage

```
ga_model_example(name, location = "googleAnalyticsR")
```
**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the model</td>
</tr>
<tr>
<td>location</td>
<td>location of model</td>
</tr>
</tbody>
</table>

**See Also**

Other GA modelling functions: `ga_model.edit`, `ga_model.load`, `ga_model.make`, `ga_model.save`, `ga_model.tweet`, `ga_model.write`, `ga_model`

---

### ga_model_load

**Load a created model**

**Description**

Load a created model

**Usage**

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>name to load model from</td>
</tr>
</tbody>
</table>

**See Also**

Other GA modelling functions: `ga_model.edit`, `ga_model.example`, `ga_model.make`, `ga_model.save`, `ga_model.tweet`, `ga_model.write`, `ga_model`

---

### 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 = graphics::plot, required_packages = NULL, description = NULL, outputShiny = shiny::plotOutput, renderShiny = shiny::renderPlot)
```
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`

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_tweet, ga_model_write, ga_model`

Examples

```r
## Not run:
get_model_data <- function(viewId, 
  date_range = c(Sys.Date()- 300, Sys.Date()), ...
){
  google_analytics(viewId, 
    date_range = date_range, 
    metrics = "sessions", 
    dimensions = "date", 
    max = -1)
}
decompose_sessions <- function(df, ...){
  decompose(ts(df$sessions, frequency = 7))
}
decomp_ga <- ga_model_make(get_model_data,
```
```r
# fetches data and outputs decomposition
ga_model(81416156, decomp_ga)

# save the model for later
model_location <- "inst/models/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 = graphics::plot,
  description = "Performs decomposition and creates a plot",
  outputShiny = shiny::plotOutput,
  renderShiny = shiny::renderPlot)

## End(Not run)
```

---

**ga_model_save**

Save a created model

**Description**

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_tweet`, `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
```r
ga_model_write(model, filepath = "ga_model.R")
```

Arguments
- `model` : The ga_model object to extract functions from to write
- `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_tweet`, `ga_model`
**ga_remarketing_build**  
*Create a remarketing audience for creation*

**Description**
Create definitions to be used within `ga_remarketing_create`

**Usage**

```r
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)

---

**ga_remarketing_create**  Create a new remarketing audience

**Description**

Create a remarketing audiences built via `ga_remarketing_build`

**Usage**

```r
ga_remarketing_create(adwordsLinkId, include, exclude = NULL,
  audienceType = c("SIMPLE", "STATE_BASED"), name = NULL)
```

**Arguments**

- **adwordsLinkId**  The adwords link to add the remarketing audience to
- **include**  A `ga4_remarketing_segment` object to include via `ga_remarketing_build`
- **exclude**  If `audienceType="STATE_BASED"`, a `ga4_remarketing_segment` object to exclude via `ga_remarketing_build`
- **audienceType**  SIMPLE or STATE_BASED
- **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: `ga_remarketing_build, 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)
```

ga_remarketing_estimate

*Estimate number of users added to the segment yesterday*

Description

Estimate number of users added to the segment yesterday

Usage

```r
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
**ga_remarketing_get**

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

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

**Value**  
Remarketing audience list

**See Also**
- About remarketing audiences
- Other remarketing 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**  
```javascript
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_unsampled | Get Unsampled Report Meta Data

**Description**

Get Unsampled Report Meta Data

**Usage**

```r
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

**See Also**

Other unsampled download functions: `ga_unsampled_download, ga_unsampled_list`

---

**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

## 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

**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",
                     downloadFile = FALSE)
```
Create or update user access to Google Analytics

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, 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**

```r
ga_users_delete(email, accountId)
```

**Arguments**

- `email`  
The email of the user to delete
- `accountId`  
The accountId that the user will be deleted from including all web properties and Views underneath.

**Details**

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

**See Also**

- [Google Documentation](#)
- Other User management functions: `ga_users_add`, `ga_users_delete_linkid`, `ga_users_list`, `ga_users_update`
Examples

```r
## 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"), 1234567)

## End(Not run)
```

gga_users_delete_linkid  

Delete users access from account, webproperty or view level

Description

The `linkId` is in the form of the `accountId/webPropertyId/viewId` colon separated from a link unique Id.

Delete user access by supplying the `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 `linkId` for that level instead. e.g. a user needs `permissions.local` to be non-NULL to be deleted at that level. The parameter `check` will do this check before deletion and throw an error if they can not be deleted. Set this to `check=FALSE` to suppress this behaviour.

If you supply more than one `linkId`, 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 `linkId`s may have failed. Check via `ga_users_list` afterwards and try to delete individual `linkId`s 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.
**ga_users_list**

**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)
g_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**

```r
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

```r
## 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)
```

Determination
This is for altering existing user access.

Usage
```
    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 via toJSON 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.

See Also

Google help article on user permissions
Other User management functions: ga_users_add, ga_users_delete_linkid, ga_users_delete, ga_users_list

Examples

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

# the update to perform
o <- list(permissions = list(local = list("EDIT")))
ga_users_update("UA-123456-1:1111222233334444",
update_object = o,
accountId = 47480439,
webPropertyId = "UA-123456-1")

## End(Not run)
```

---

**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**

```r
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

```r
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`, `ga_webproperty_list`

Examples

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

## End(Not run)
```
ga_webproperty_list

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: `ga_account_list, ga_accounts, ga_view_list, ga_view, ga_webproperty`

Examples

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

googleAnalyticsR

Library for getting Google Analytics data into R

Description

Follow the online documentation here: [https://code.markedmondson.me/googleAnalyticsR/](https://code.markedmondson.me/googleAnalyticsR/)

Details

You may wish to set the below environment arguments for easier authentication

```
GA_CLIENT_ID GA_CLIENT_SECRET GA_WEB_CLIENT_ID GA_WEB_CLIENT_SECRET GA_AUTH_FILE
```
google_analytics  Get Google Analytics v4 data

Description

Fetch Google Analytics data using the v4 API. For the v3 API use google_analytics_3. See website help for lots of examples: 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  viewId of data 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 dimensions 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.
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. See details in make_ga_4_req

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.

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 gar_cache_setup function. This can be useful when running RMarkdown reports using data. To empty the cache use gar_cache_empty.
See Also

Other GAv4 fetch functions: fetch_google_analytics_4_slow, fetch_google_analytics_4, make_ga_4_req

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
google_analytics(1234567, date_range = c("30daysAgo", "yesterday"), metrics = "sessions")
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", "WALK"),
max_results = 100, multi_account_batching = FALSE, 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.
- `multi_account_batching`: If TRUE then multiple id’s are fetched together. Not compatible with samplingLevel="WALK" or max_results>10000.
- `type`: `ga` = Google Analytics v3; `mcf` = Multi-Channel Funels.

Value

For one id a data.frame of data, with meta-data in attributes. For multiple id’s, a list of dataframes.

See Also

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

Examples

```r
## 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

```r
count_list <- ga_account_list()
ga_id <- account_list$viewId[1]
```

## get a list of what metrics and dimensions you can use

```r
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

```r
gadata <- google_analytics_3(id = ga_id,
                          start="2015-08-01", end="2015-08-02",
                          metrics = c("sessions", "bounceRate"),
                          dimensions = c("source", "medium"))
```

## multi accounts, pass character vector of viewIds

## outputs a list of dataframes, named after the viewId

```r
multi_gadata <- google_analytics_3(id = c("123456", "9876545", "765432"),
                                    start="2015-08-01", end="2015-08-02",
                                    metrics = c("sessions", "bounceRate"),
                                    dimensions = c("source", "medium"))
```

## if more than 10000 rows in results, auto batching

## example is setting lots of dimensions to try and create big sampled data

```r
batch_gadata <- google_analytics_3(id = ga_id,
                                    start="2014-08-01", end="2015-08-02",
                                    metrics = c("sessions", "bounceRate"),
                                    dimensions = c("source", "medium",
                                    "landingPagePath", "hour","minute"),
                                    max=99999999)
```

## mitigate sampling by setting samplingLevel="WALK"

## this will send lots and lots of calls to the Google API limits, beware

```r
walk_gadata <- google_analytics_3(id = ga_id,
                                  start="2014-08-01", end="2015-08-02",
                                  metrics = c("sessions", "bounceRate"),
                                  dimensions = c("source", "medium", "landingPagePath"),
                                  max=99999999, samplingLevel="WALK")
```

## multi-channel funnels set type="mcf"

```r
mcf_gadata <- google_analytics_3(id = ga_id,
                                 start="2015-08-01", end="2015-08-02",
                                 metrics = c("totalConversions"),
                                 dimensions = c("sourcePath"),
                                 type="mcf")
```

## reach meta-data via attr()

```r
attr(gadata, "profileInfo")
```
### google_analytics_bq

`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)`

`attr(gadata, "dateRange")`

## End(Not run)

---

**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.
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

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 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

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**

See Also

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

Other v4 cohort functions: cohortGroup, cohort_dimension_check, cohort_metric_check, cohort

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']

## 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'),
                                 cohort = cohort_ltv,
                                 metrics = c('cohortTotalUsers', 'lifetimeValue'))
```

make_ga_4_req

```r
dimensions = c('cohort', "acquisitionTrafficChannel"),
cohorts = cohort_ltv,
metrics = c("cohortGoalCompletionsPerUserWithLifetimeCriteria")
```

## End(Not run)

---

### make_ga_4_req

**Make a Google Analytics v4 API fetch**

### Description

This function constructs the Google Analytics API v4 call to be called via `fetch_google_analytics_4`

### Usage

```r
make_ga_4_req(viewId, date_range = NULL, metrics = NULL,
dimensions = NULL, dim_filters = NULL, met_filters = NULL,
filtersExpression = NULL, order = NULL, segments = NULL,
pivots = NULL, cohorts = NULL, pageToken = 0, pageSize = 1000,
samplingLevel = c("DEFAULT", "SMALL", "LARGE"), metricFormat = NULL,
histogramBuckets = NULL)
```

### Arguments

- **viewId**
  - viewId of data 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 dimensions 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 dimensions 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`
- **pageToken**
  - Where to start the data fetch
- **pageSize**
  - How many rows to fetch. Max 100000 each batch.
- **samplingLevel**
  - Sample level
make_ga_4_req

metricFormat  If supplying calculated metrics, specify the metric type

histogramBuckets
For numeric dimensions such as hour, a list of buckets of data. See details in
make_ga_4_req

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:

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 met-
rics. 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+

See Also

Other GA4 fetch functions: fetch_google_analytics_4_slow, fetch_google_analytics_4,
google_analytics

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']

ga_req1 <- make_ga_4_req(ga_id,
    date_range = c("2015-07-30","2015-10-01"),
    dimensions=c('source','medium'),
    metrics = c('sessions'))

ga_req2 <- make_ga_4_req(ga_id,
    date_range = c("2015-07-30","2015-10-01"),
    dimensions=c('source','medium'),
    metrics = c('users'))

fetch_google_analytics_4(list(ga_req1, ga_req2))

## End(Not run)

---

**meta**

*Google Analytics API metadata*

### Description

This is a local copy of the data provided by ga_meta

### 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

[https://developers.google.com/analytics/devguides/reporting/core/dimsmets](https://developers.google.com/analytics/devguides/reporting/core/dimsmets)
**met_filter**

*Make a metric filter object*

### 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()

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

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

## construct filter objects

```r
fc2 <- filter_clause_ga4(list(df, df2), operator = "AND")
fck <- filter_clause_ga4(list(df, df2), operator = "AND")
```

## make v4 request

```r
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 = df2,
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
- `type`: metric or dimension
- `subType`: Limit selections to those relevant
- `default`: The default selected choice. First element if NULL

**Details**

Call via `shiny::callModule(multi_select,"your_id")`
multi_selectUI

Value

the selected variable

See Also

Other Shiny modules: authDropdownUI, authDropdown, multi_selectUI

---

multi_selectUI  multi_select UI [Shiny Module]

Description

Shiny Module for use with multi_select

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Shiny id</td>
</tr>
<tr>
<td>label</td>
<td>label</td>
</tr>
<tr>
<td>multiple</td>
<td>multiple select</td>
</tr>
<tr>
<td>width</td>
<td>width of select</td>
</tr>
</tbody>
</table>

Details

Create a Google Analytics variable selector

Value

Shiny UI

See Also

Other Shiny modules: authDropdownUI, authDropdown, multi_select
**order_type**  
*Make an OrderType object*

**Description**  
Make an OrderType object

**Usage**  

```r
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**  

```r
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.
pivot_ga4

Details

If maxGroupCount is set to -1 returns all groups.

Value

pivot object of class pivot_ga4 for use in 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
count_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'),
                          metrics = c('sessions'),
                          pivots = list(pivme))

## End(Not run)
```
segmentBuilder

Create a GA4 Segment Builder

Description

Shiny Module for use with segmentBuilderUI

Usage

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)
```
segmentBuilderUI

Create a GAv4 Segment Builder

Description
Shiny Module for use with segmentBuilder

Usage
segmentBuilderUI(id)

Arguments

id Shiny id

Value
Shiny UI for use in app

Examples

## Not run:

```r
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)
```
### segment_define

**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

**Make a 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, operator = c("REGEXP", "BEGINS_WITH", "ENDS_WITH", "PARTIAL", "EXACT", "IN_LIST", "NUMERIC_LESS_THAN", "NUMERIC_GREATER_THAN", "NUMERIC_BETWEEN", "LESS_THAN", "GREATER_THAN", "EQUAL", "BETWEEN"), 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**: [dim] What the name shall compare to
- **caseSensitive**: [dim] Whether to be case sensitive
- **minComparisonValue**: [dim] Minimum comparison values for BETWEEN
- **maxComparisonValue**: Max comparison value for BETWEEN operator
- **scope**: [met] Scope of the metric value
- **comparisonValue**: [met] What the name shall compare to
- **matchType**: If used in sequence segment, what behaviour

Value

An 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

```r
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. gaid:-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_sequence

*Make sequenceSegment*

**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**

segment_vector_sequence(segment_elements, firstStepMatch = FALSE)

**Arguments**

- `segment_elements` a list of OR lists of segment elements
- `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
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

**Topic datasets**
- meta, 82

authDropdown, 4, 5, 85
authDropdownUI, 4, 5, 85
bqr_list_projects, 78

cohort, 79
cohort_dimension_check, 79
cohort_metric_check, 79
cohortGroup, 79
dim_filter, 5, 9, 72, 80, 83

fetch_google_analytics_4, 7, 8, 74, 80, 81
fetch_google_analytics_4_slow, 7, 8, 74, 80

filter_clause_ga4, 6, 9, 72, 80, 83

ga_account_list, 10, 11, 69–71
gas_accounts, 10, 11, 69–71

gas_adwords, 12, 13, 14
gas_adwords_add_linkid, 12, 12, 13, 14
gas_adwords_delete_linkid, 12, 13, 13, 14
gas_adwords_list, 12, 13, 14
gas_aggregate, 15
gas_allowed_metric_dim, 16
gas_auth, 16
gas_cache_call, 18
gas_clientid_activity, 19, 21, 23, 24
gas_clientid_activity_unnest, 19, 20, 23, 24
gas_clientid_deletion, 19, 21, 22, 24
gas_clientid_hash, 19, 21, 23, 23
gas_custom_datasource, 24, 25, 27, 29
gas_custom_upload, 24, 25, 27, 29
gas_custom_upload_delete, 24, 25, 26, 27, 29
gas_custom_upload_file, 24, 25, 27, 29
gas_custom_upload_list, 24, 25, 27, 28
gas_custom_vars, 29, 30–32
gas_custom_vars_create, 29, 30, 31, 32
gas_custom_vars_list, 29, 30, 31, 32
gas_custom_vars_patch, 29–31, 32
gas_experiment, 33, 34, 35, 37, 39, 41, 59
gas_experiment_list, 33, 34, 35, 37, 39, 41, 59
gas_filter, 34, 38, 39, 42
gas_filter_add, 33, 34, 35, 37, 39, 41, 59
gas_filter_apply_to_view, 33–35, 37, 39, 41, 59
gas_filter_delete, 35, 38, 39, 42
gas_filter_list, 35, 38, 39, 42
gas_filter_update, 33–35, 37, 39, 41, 59
gas_filter_update_filter_link, 33–35, 37, 39, 40, 59
gas_filter_view, 35, 38, 39, 41, 42
gas_filter_view_list, 35, 38, 39, 42, 42
gas_goal, 43, 44, 46, 47
gas_goal_add, 43, 43, 46, 47
gas_goal_list, 43, 44, 46, 47
gas_goal_update, 43, 44, 46, 46
gas_meta, 48, 82
gas_model, 48, 49–51, 53, 54
gas_model_edit, 48, 49, 50, 51, 53, 54
gas_model_example, 48, 49, 49, 50, 51, 53, 54
gas_model_load, 48–50, 50, 51, 53, 54
gas_model_make, 48–50, 50, 53, 54
gas_model_save, 48–51, 52, 53, 54
gas_model_tweet, 48–51, 53, 53, 54
gas_model_write, 48–51, 53, 54
gas_remarketing_build, 55, 56, 58, 59
gas_remarketing_create, 55, 56, 58, 59
gas_remarketing_estimate, 55, 56, 57, 58, 59
gas_remarketing_get, 55–58, 58, 59
gas_remarketing_list, 55, 56, 58, 59
gas_segment_list, 33–35, 37, 39, 41, 59
gas_unsampled, 60, 61, 62

96
ga_unsampled_download, 60, 60, 62
ga_unsampled_list, 60, 61, 62
ga_users_add, 63, 64, 66–68
ga_users_delete, 63, 64, 66–68
ga_users_delete_linkid, 63, 64, 65, 67, 68
ga_users_list, 63–66, 66, 68
ga_users_update, 63, 64, 66, 67, 67
ga_view, 10, 11, 68, 69–71
ga_view_list, 10, 11, 69, 69, 70, 71
ga_webproperty, 10, 11, 69, 70, 71
ga_webproperty_list, 10, 11, 69, 70, 71
gar_auth, 16
gar_auth_service, 16
gar_cache_empty, 73
gar_cache_setup, 73
gar_set_client, 17, 73
goog[le_analytics], 7, 8, 11, 72, 73, 74, 81
google_analytics_3, 72, 74
goog[le_analytics]_4 (goog[le_analytics]), 72
google_analytics_bq, 77
goog[leAnalyticsR], 71
goog[leAnalyticsR]-package
  (goog[leAnalyticsR]), 71
make_cohort_group, 72, 78, 80
make_ga_4_req, 7–9, 73, 74, 80, 81
met_filter, 6, 9, 72, 80, 83
meta, 72, 80, 82
multi_select, 4, 5, 84, 85
multi_selectUI, 4, 5, 84, 85, 85
order_type, 72, 80, 86
pivot_ga4, 72, 80, 86
segment_define, 90, 90, 91, 92, 94, 95
segment_element, 90, 90, 92, 94, 95
segment_ga4, 72, 80, 90, 91, 91, 94, 95
segment_vector_sequence, 90–92, 94, 94, 95
segment_vector_simple, 90–92, 94, 95, 95
segmentBuilder, 88, 89
segmentBuilderUI, 88, 89
Startup, 17
Sys.setenv, 17
toJSON, 67