

Package ‘measurementProtocol’

March 4, 2023

Type Package

Version 0.1.1

Title Send Data from R to the Measurement Protocol

Description Send server-side tracking data from R.

The Measurement Protocol version 2

<<https://developers.google.com/analytics/devguides/collection/protocol/ga4>>
allows sending HTTP tracking events from R code.

URL <https://code.markedmondson.me/measurementProtocol/>

BugReports <https://github.com/MarkEdmondson1234/measurementProtocol>

Depends R (>= 3.3.0)

Imports assertthat (>= 0.2.0), cli, httr (>= 1.3.1), jsonlite (>= 1.5), rappdirs (>= 0.3.3), stats, utils

License MIT + file LICENSE

RoxygenNote 7.1.1

Config/testthat/edition 3

Encoding UTF-8

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

NeedsCompilation no

Author Mark Edmondson [aut, cre] (<<https://orcid.org/0000-0002-8434-3881>>),
Sunholo Ltd [cph]

Maintainer Mark Edmondson <r@sunholo.com>

Repository CRAN

Date/Publication 2023-03-04 16:30:02 UTC

R topics documented:

| | |
|---------------|---|
| mp_cid | 2 |
| mp_event | 3 |
| mp_event_item | 3 |

| | |
|-------------------------|----|
| mp_opt_in | 5 |
| mp_parse_json | 5 |
| mp_send | 8 |
| mp_trackme | 10 |

| | |
|--------------|-----------|
| Index | 13 |
|--------------|-----------|

| | |
|--------|------------------------------------|
| mp_cid | <i>Generate a random client_id</i> |
|--------|------------------------------------|

Description

This has a random number plus a timestamp

Usage

```
mp_cid(seed = NULL)
```

Arguments

| | |
|------|---|
| seed | If you set a seed, then the random number will be the same for each value |
|------|---|

Value

A string suitable as an Id with a random number plus a timestamp delimited by a period.

See Also

Other Measurement Protocol functions: [mp_event_item\(\)](#), [mp_event\(\)](#), [mp_send\(\)](#)

Examples

```
# random Id
mp_cid()

# fix the random number (but not the timestamp)
mp_cid(1)
```

| | |
|----------|--|
| mp_event | <i>Create a Measurement Protocol Event</i> |
|----------|--|

Description

[Experimental] This creates an event to send via [mp_send](#)

Usage

```
mp_event(name, params = NULL, items = NULL)
```

Arguments

| | |
|--------|--|
| name | The event name to send in |
| params | Optional event parameters sent in as a named list |
| items | Optional items created via mp_event_item |

Value

An mp_event object

See Also

Other Measurement Protocol functions: [mp_cid\(\)](#), [mp_event_item\(\)](#), [mp_send\(\)](#)

Examples

```
mp_event("custom_event")
mp_event("custom_event", params = list(my_param = "SUPER"))
```

| | |
|---------------|--|
| mp_event_item | <i>Create an Measurement Protocol Item Property for an Event</i> |
|---------------|--|

Description

[Experimental] Some events work with item properties

Usage

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

Arguments

| | |
|---------------|-------------|
| item_id | Item ID |
| item_name | Item Name |
| coupon | Coupon |
| discount | Discount |
| affiliation | Affiliation |
| item_brand | Brand |
| item_category | Category |
| item_variant | Variant |
| price | Price |
| currency | Currency |

Value

An mp_event_item object

See Also

Other Measurement Protocol functions: [mp_cid\(\)](#), [mp_event\(\)](#), [mp_send\(\)](#)

Examples

```
# one item  
mp_event_item(item_name = "jeggings",  
              price = 8.88,  
              item_variant = "Black")  
  
# many items in a list  
items <- list(  
  mp_event_item(item_id = "SKU_12345",  
                price = 9.99,  
                item_brand = "Gucci"),
```

```

mp_event_item(item_name = "jeggings",
              price = 8.88,
              item_variant = "Black"))

# construct an event with its own fields
mp_event("add_payment_info",
        params = list(coupon = "SUMMER_FUN",
                      payment_type = "Credit Card",
                      value = 7.77,
                      currency = "USD"),
        items = items)

```

| | |
|-----------|---|
| mp_opt_in | <i>Tracking opt-in for this package</i> |
|-----------|---|

Description

This is the opt-in function for this package, using [mp_trackme](#)

Usage

```
mp_opt_in()
```

Value

No return value, called for side effects

| | |
|---------------|--|
| mp_parse_json | <i>Parse out objects into the Measurement Protocol v2 format for sending</i> |
|---------------|--|

Description

This function helps take HTTP events and rearranges its structure so it will work in a MP measurement protocol hit. This enables HTTP events from say Pub/Sub to be translated into MP hits.

Usage

```

mp_parse_json(
  json,
  name_f,
  params_f = NULL,
  items_f = NULL,
  client_id_f = NULL,
  user_id_f = NULL,
  user_properties_f = NULL
)

```

```
mp_parse_gtm(json)
```

```
mp_pubsub(pubsub_body)
```

Arguments

| | |
|-------------------|--|
| json | The location of a json file or a json string or an R list that has been parsed from json via <code>jsonlite::fromJSON</code> |
| name_f | The function that extracts the event name out of json |
| params_f | An optional function that extracts parameters for the event from json |
| items_f | An optional function that extracts e-commerce items from json. Must return a mp_event_item object. you may not need this if the <code>params_f</code> includes parsing of e-commerce items |
| client_id_f | An optional function to extract the client.id. You will need to supply <code>cid</code> though if using downstream in <code>mp_send</code> so it usually is necessary |
| user_id_f | Optionally include a function that will parse out <code>user_id</code> |
| user_properties_f | Optionally include a function that will parse out user properties |
| pubsub_body | The <code>req\$postBody</code> of a plumber request |

Details

The passed in functions should return NULL if they don't find any entries

Value

An `mp_parse_json` object that is a list of an `mp_event` object, and user fields including `client.id`, `user.id` and user properties

The Pub/Sub message "data" attribute unencoded into a json string

Examples

```
demo_json <- system.file("example", "pubsub-ga4.json", package = "measurementProtocol")
demo_list <- jsonlite::fromJSON(demo_json)

# extract the event_name
name_f <- function(x) x[["event_name"]]

# extract client_id
client_id_f <- function(x) x[["client_id"]]

# extract user_id
user_id_f <- function(x) x[["user_id"]]

# simple event
```

```

mp_parse_json(demo_list,
              name_f,
              client_id_f = client_id_f,
              user_id_f = user_id_f)

# params could be assumed to be everything not a event_name of client_id
# also not allowed any starting with reserved 'ga_'
params_f <- function(x){
  x_names <- names(x)[grepl("^x-", names(x))]
  ga_names <- names(x)[grepl("^ga_", names(x))]
  x[setdiff(names(x), c("client_id", "user_id", "event_name", x_names, ga_names))]
}

# parse including params (could include items as well)
parsed_event <- mp_parse_json(demo_list,
                              name_f,
                              params_f = params_f,
                              client_id_f = client_id_f,
                              user_id_f = user_id_f)

parsed_event

# sending to a debug endpoint
# preferably set this in .Renviron
Sys.setenv(MP_SECRET="MY_SECRET")

# replace with your GA4 settings
my_measurement_id <- "G-1234"
my_connection <- mp_connection(my_measurement_id)
mp_send(parsed_event$mp_event,
        client_id = parsed_event$user$client_id,
        user_id = parsed_event$user$user_id,
        user_properties = parsed_event$user$user_properties,
        connection = my_connection,
        debug_call = TRUE)

# mp_parse_gtm internally uses functions demonstrated with mp_parse_json
pubsub_event <- mp_parse_gtm(demo_json)

mp_send(pubsub_event$mp_event,
        client_id = pubsub_event$user$client_id,
        user_id = pubsub_event$user$user_id,
        user_properties = pubsub_event$user$user_properties,
        connection = my_connection,
        debug_call = TRUE)

## Not run:

## Send forward a measurement protocol hit
## @post /gtm
## @serializer unboxedJSON
## @parser json

```

```

function(req, res, ga_id) {

  pubsub_data <- mp_pubsub_parse(req$postBody)

  parsed <- mp_parse_gtm(pubsub_data)

  my_connection <- mp_connection(ga_id)

  mp_send(parsed$mp_event,
          client_id = parsed$user$client_id,
          user_id = parsed$user$user_id,
          user_properties = parsed$user$user_properties,
          connection = my_connection)

  "OK"
}

## End(Not run)

```

mp_send

Make a Measurement Protocol v2 request

Description

[Experimental] Create a server side call to Google Analytics 4 via its Measurement Protocol. Use [mp_connection](#) to set up the Measurement Protocol connections to pass to [mp_send](#). If using Google Tag Manager Server-Side, you can also set up a custom endpoint.

Usage

```

mp_send(
  events,
  client_id,
  connection,
  user_id = NULL,
  debug_call = FALSE,
  timestamp_micros = NULL,
  user_properties = NULL,
  non_personalized_ads = TRUE
)

mp_connection(
  measurement_id,
  api_secret = Sys.getenv("MP_SECRET"),
  endpoint = NULL,
  preview_header = NULL
)

```

Arguments

| | |
|----------------------|--|
| events | The events to send |
| client_id | The client_id to associate with the event |
| connection | The connection details created by mp_connection |
| user_id | Optional. Unique id for the user |
| debug_call | Send hits to the Google debug endpoint to validate hits. |
| timestamp_micros | Optional. A Unix timestamp (in microseconds) for the time to associate with the event. |
| user_properties | Optional. The user properties for the measurement sent in as a named list. |
| non_personalized_ads | Optional. Set to true to indicate these events should not be used for personalized ads. |
| measurement_id | The measurement ID associated with a stream |
| api_secret | The secret generated in the GA4 UI - by default will look for environment arg MP_SECRET |
| endpoint | If NULL will use Google default, otherwise set to the URL of your Measurement Protocol custom endpoint |
| preview_header | Only needed for custom endpoints. The X-Gtm-Server-Preview HTTP Header found in your GTM debugger |

Details

Create an API secret via Admin > Data Streams > choose your stream > Measurement Protocol > Create
 To see event parameters, create custom fields in your GA4 account first, to see them in your reports 24hrs after you send them in with this function via Custom definitions > Create custom dimensions - dimension name will be how it looks like in the reports, event parameter will be the parameter you have sent in with the event.

user_id can be used for [cross-platform analysis](#)

timestamp_micros should only be set to record events that happened in the past. This value can be overridden via user_property or event timestamps. Events can be backdated up to 48 hours. Note microseconds, not milliseconds.

user_properties - describe segments of your user base, such as language preference or geographic location. See [User properties](#)

Ensure you also have user permission as specified in the [feature policy](#)

Invalid events are silently rejected with a 204 response, so use debug_call=TRUE to validate your events first.

Value

TRUE if successfully sent the hit. If debug_call=TRUE then the JSON response from the debugger endpoint

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

An mp_connection class object

See Also

[Measurement Protocol \(Google Analytics 4\)](#)

Other Measurement Protocol functions: [mp_cid\(\)](#), [mp_event_item\(\)](#), [mp_event\(\)](#)

Examples

```
# preferably set this in .Renvirom
Sys.setenv(MP_SECRET="MY_SECRET")

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

my_connection <- mp_connection(my_measurement_id)

a_client_id <- 123.456
event <- mp_event("an_event")
mp_send(event, a_client_id, my_connection, debug_call = TRUE)

# multiple events at same time in a batch
another <- mp_event("another_event")

mp_send(list(event, another),
        a_client_id,
        my_connection,
        debug_call = TRUE)

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

## End(Not run)

# custom GTM server side endpoint
my_custom_connection <- mp_connection(
  my_measurement_id,
  endpoint = "https://gtm.example.com",
  preview_header = "ZW52LTV80WdPOExNWFkYjA0Njk4NmQ="
)
```

Description

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

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

Usage

```
mp_trackme(package)

mp_trackme_event(
  package,
  debug_call = FALSE,
  say_hello = NULL,
  opt_in_function = NULL
)
```

Arguments

| | |
|------------------------------|--|
| <code>package</code> | The package name |
| <code>debug_call</code> | Set as a debug event to see what would be sent |
| <code>say_hello</code> | If you want to add your own custom message to the event sent, add it here! |
| <code>opt_in_function</code> | The name of the function for a user to opt-in |

Details

Running this function will send a Measurement Protocol hit via [mp_send](#) only if the cache file is present

Value

No return value, called for side effects

Examples

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

# this only works with a valid opt-in file present
mp_trackme_event("googleAnalyticsR")

# see what data is sent
mp_trackme_event("googleAnalyticsR", debug_call=TRUE)

# add your own message!
```

```
mp_trackme_event("googleAnalyticsR",
                 debug_call = TRUE,
                 say_hello = "err hello Mark")

# placed in .onAttach with function name
.onAttach <- function(libname, pkgname){
  measurementProtocol::mp_trackme_event(pkgname, opt_in_function = "mp_opt_in")
}
```

Index

* Measurement Protocol functions

- [mp_cid](#), [2](#)
 - [mp_event](#), [3](#)
 - [mp_event_item](#), [3](#)
 - [mp_send](#), [8](#)

- [mp_cid](#), [2](#), [3](#), [4](#), [10](#)
- [mp_connection](#), [8](#), [9](#)
- [mp_connection](#) ([mp_send](#)), [8](#)
- [mp_event](#), [2](#), [3](#), [4](#), [10](#)
- [mp_event_item](#), [2](#), [3](#), [3](#), [6](#), [10](#)
- [mp_opt_in](#), [5](#)
- [mp_parse_gtm](#) ([mp_parse_json](#)), [5](#)
- [mp_parse_json](#), [5](#)
- [mp_pubsub](#) ([mp_parse_json](#)), [5](#)
- [mp_send](#), [2-4](#), [8](#), [8](#), [11](#)
- [mp_trackme](#), [5](#), [10](#)
- [mp_trackme_event](#) ([mp_trackme](#)), [10](#)