Piwik PRO Analytics Suite Documentation

Release 16.4

Piwik PRO

May 05, 2022

Contents

1 Analytics **3**
- 1.1 Columns . . . 3
- 1.2 Integrations . . . 9
- 1.3 HTTP API . . . 16
- 1.4 Object management API . . . 16
- 1.5 Metrics Mapping . . . 16

2 Data Collection **21**
- 2.1 Web . . . 21
- 2.2 Mobile . . . 69
- 2.3 API . . . 93
- 2.4 Other integrations . . . 93

3 Audience Manager **101**
- 3.1 Profile data . . . 102
- 3.2 JavaScript API . . . 103
- 3.3 Form Tracker . . . 108
- 3.4 Public HTTP API . . . 111
- 3.5 Authorized HTTP API . . . 111

4 Consent Manager **113**
- 4.1 Custom consent form . . . 113
- 4.2 JavaScript API . . . 113

5 Tag Manager **123**
- 5.1 Authorized HTTP API . . . 123
- 5.2 Custom data layer name . . . 123
- 5.3 Content Security Policy (CSP) . . . 126
- 5.4 Custom popup template implementation examples . . . 130
- 5.5 Skip link tracking with *data-disable-delay* attribute . . . 142

6 Administration **145**
- 6.1 Getting started . . . 145
- 6.2 Access Control API . . . 150
- 6.3 Apps API . . . 150
- 6.4 Audit log API . . . 150
- 6.5 Meta Sites API . . . 150

6.6 Modules API . . . 150
6.7 Collecting & Processing Pipeline Settings API . . . 150
6.8 User Groups API . . . 150
6.9 Users API . . . 150

7 Glossary **151**

Index **153**

Changelog

CHAPTER 1

Analytics

1.1 Columns

This article documents core columns available in the *HTTP API*. Additional columns may become available through *Integrations*.

Note: Each column listed in this document defines a *Scope* attribute. If you request a query that includes at least one column which requires *event* scope, the entire query will be calculated using events, instead of sessions. This might distort some custom metrics such as averages of a *session* dimension (e.g. average session time).

1.1.1 Metrics

The table below lists core metrics that may be used in queries. Additional metrics may be created using dimension transformations.

Table 1: Base Metrics

Metric Name	Column ID	Scope	Type
Events	events	session	int
Consent form impressions	consent_form_impressions	event	int
Consent form clicks	consent_form_clicks	event	int
First consents	consents_first	event	int
Changed consents	consents_changed	event	int
Full consents	consents_full	event	int
Any consents	consents_any	event	int
No consents	consents_none	event	int
No decisions	consents_no_decision	event	int
Analytics consents	consents_analytics	event	int
A/B testing personalization consents	consents_ab_testing_personalization	event	int

Continued on next page

Table 1 – continued from previous page

Metric Name	Column ID	Scope	Type
Conversion tracking consents	consents_conversion_tracking	event	int
Marketing automation consents	consents_marketing_automation	event	int
Remarketing consents	consents_remarketing	event	int
User feedback consents	consents_user_feedback	event	int
Custom consent 1	consents_custom_1	event	int
Page views	page_views	session	int
Unique page views	unique_page_views	session	int
Entries	entries	session	int
Exits	exits	session	int
Bounces	bounces	session	int
Sessions	sessions	session	int
Visitors	visitors	session	int
% of returning visitors	returning_visitors_rate	session	float
Users	users	session	int
Visitor IPs	visitor_ips	session	int
Outlinks	outlinks	session	int
Unique outlinks	unique_outlinks	session	int
Downloads	downloads	session	int
Unique downloads	unique_downloads	session	int
Searches	searches	session	int
Unique searches	unique_searches	session	int
Custom events	custom_events	session	int
Unique custom events	unique_custom_events	session	int
Content impressions	content_impressions	session	int
Unique content impressions	unique_content_impressions	session	int
Content interactions	content_interactions	session	int
Unique content interactions	unique_content_interactions	session	int
Goal conversions	goal_conversions	session	int
Unique goal conversions	unique_goal_conversions	session	int
Ecommerce conversions	ecommerce_conversions	session	int
Ecommerce abandoned carts	ecommerce_abandoned_carts	session	int
Unique purchases	unique_purchases	event	int
Entry rate	entry_rate	session	float
Exit rate	exit_rate	session	float
Exit rate events	exit_rate_events	session	float
Bounce rate	bounce_rate	session	float
Bounce rate	bounce_rate_events	session	float
Content interaction rate	content_interaction_rate	session	float
Goal conversion rate	goal_conversion_rate	session	float
Ecommerce conversion rate	ecommerce_conversion_rate	session	float
Events per session	events_per_session	session	float

1.1.2 Dimensions

The table below lists core dimensions that may be used in queries.

Note: “Database type” column presents the type of source column of the dimension (in case of enum - type of the ID, in case of dynamic dimensions - not applicable).

Table 2: Base Dimensions

Dimension Name	Column ID	Scope	Type	Database Type	Nullab
Visitor ID	visitor_id	session	hex	uint64	False
User ID	user_id	session	str	string	False
Cookie ID	cookie_id	session	hex	uint64	False
Returning visitor	visitor_returning	session	[int, str]	uint8	False
Session number	visitor_session_number	session	int	uint16	False
Days since last session	visitor_days_since_last_session	session	int	uint16	True
Days since first session	visitor_days_since_first_session	session	int	uint16	True
Days since order	visitor_days_since_order	session	int	uint16	True
Events in session	session_total_events	session	int	uint16	False
Session time	session_total_time	session	int	uint32	False
Page views in session	session_total_page_views	session	int	uint16	False
Outlinks in session	session_total_outlinks	session	int	uint16	False
Downloads in session	session_total_downloads	session	int	uint16	False
Site searches in session	session_total_site_searches	session	int	uint16	False
Custom events in session	session_total_custom_events	session	int	uint16	False
Content impressions in session	session_total_content_impressions	session	int	uint16	False
Content interactions in session	session_total_content_interactions	session	int	uint16	False
Goal conversions in session	session_total_goal_conversions	session	int	uint16	False
Ecommerce conversions in session	session_total_ecommerce_conversions	session	int	uint16	False
Abandoned carts in session	session_total_abandoned_carts	session	int	uint16	False
Unique page views in session	session_unique_page_views	session	int	uint16	False
Unique outlinks in session	session_unique_outlinks	session	int	uint16	False
Unique downloads in session	session_unique_downloads	session	int	uint16	False
Unique site searches in session	session_unique_searches	session	int	uint16	False
Unique custom events in session	session_unique_custom_events	session	int	uint16	False
Unique content impressions in session	session_unique_content_impressions	session	int	uint16	False
Unique content interactions in session	session_unique_content_interactions	session	int	uint16	False
Goals converted in session	session_goals	session	array	array of int32	False
Shopping stage	session_ecommerce_status	session	[int, str]	uint8	False
Source	source	session	str	string	False
Medium	medium	session	str	string	False
Source/Medium	source_medium	session	str	string	False
Keyword	keyword	session	str	string	False
Channel	referrer_type	session	[int, str]	uint8	False
Referrer URL	referrer_url	session	str	string	False
Campaign name	campaign_name	session	str	string	False
Campaign ID	campaign_id	session	str	string	False
Campaign content	campaign_content	session	str	string	False
Google Click ID	campaign_gclid	session	str	string	True
Operating system	operating_system	session	[str, str]	string(3)	True
Operating system version	operating_system_version	session	str	string	False
Browser engine	browser_engine	session	str	string	False
Browser name	browser_name	session	[str, str]	string(2)	True
Browser version	browser_version	session	str	string	False
Browser language	browser_language_iso639	session	[str, str]	string(2)	True
Browser fingerprint	browser_fingerprint	session	int	uint64	False
Device type	device_type	session	[int, str]	uint8	True
Device brand	device_brand	session	[str, str]	string(2)	True
Device model	device_model	session	str	string	False

Table 2 – continued from previous page

Dimension Name	Column ID	Scope	Type	Database Type	Nullab
Resolution	resolution	session	str	string	True
Resolution width	resolution_width	session	int	uint16	True
Resolution height	resolution_height	session	int	uint16	True
PDF plugin	plugin_pdf	session	int(0,1)	uint8	False
Flash plugin	plugin_flash	session	int(0,1)	uint8	False
Java plugin	plugin_java	session	int(0,1)	uint8	False
Director plugin	plugin_director	session	int(0,1)	uint8	False
QuickTime plugin	plugin_quicktime	session	int(0,1)	uint8	False
RealPlayer plugin	plugin_realplayer	session	int(0,1)	uint8	False
Windows Media Player plugin	plugin_windowsmedia	session	int(0,1)	uint8	False
Gears plugin	plugin_gears	session	int(0,1)	uint8	False
Silverlight plugin	plugin_silverlight	session	int(0,1)	uint8	False
Cookie support	plugin_cookie	session	int(0,1)	uint8	False
Continent	location_continent_iso_code	session	[str, str]	string(2)	True
Country	location_country_name	session	[str, str]	string	True
Subdivision	location_subdivision_1_name	session	[str, str]	string	True
Subdivision 2	location_subdivision_2_name	session	[str, str]	string	True
City	location_city_name	session	[int, str]	string	True
Designated market area	location_metro_code	session	[str, str]	string(3)	True
Latitude	location_latitude	session	float	float64	True
Longitude	location_longitude	session	float	float64	True
Provider	location_provider	session	str	string	False
Organization	location_organization	session	str	string	False
Session exit URL	session_exit_url	session	str	string	False
Session exit title	session_exit_title	session	str	string	False
Session entry URL	session_entry_url	session	str	string	False
Session entry title	session_entry_title	session	str	string	False
Session second URL	session_second_url	session	str	string	False
Session second title	session_second_title	session	str	string	False
Session bounce	is_bounce	session	int(0,1)	uint8	False
Event ID	event_id	event	int	uint64	False
Session ID	session_id	session	int	uint64	False
Exit view	is_exit	event	int(0,1)	uint8	False
Entry view	is_entry	event	int(0,1)	uint8	False
Event type	event_type	event	[int, str]	uint8	False
Page URL	event_url	event	str	string	False
Page title	event_title	event	str	string	False
Outlink URL	outlink_url	event	str	string	False
Download URL	download_url	event	str	string	False
Search keyword	search_keyword	event	str	string	False
Search category	search_category	event	str	string	False
Search results count	search_results_count	event	int	uint16	True
Custom event category	custom_event_category	event	str	string	False
Custom event action	custom_event_action	event	str	string	False
Custom event name	custom_event_name	event	str	string	False
Custom event value	custom_event_value	event	float	float64	True
Content name	content_name	event	str	string	False
Content piece	content_piece	event	str	string	False
Content target	content_target	event	str	string	False

Table 2 – continued from previous page

Dimension Name	Column ID	Scope	Type	Database Type	Nullab
Previous page view URL	previous_event_url	event	str	string	False
Previous page view title	previous_event_title	event	str	string	False
Next page view URL	next_event_url	event	str	string	False
Next page view title	next_event_title	event	str	string	False
Event index	event_index	event	int	uint16	False
Page view index	page_view_index	event	int	uint16	True
Time on page	time_on_page	event	int	uint32	True
Page generation time	page_generation_time	event	float	float64	True
Goal name	goal_id	event	[int, str]	int32	True
Goal revenue	goal_revenue	event	float	float64	True
Lost revenue	lost_revenue	event	float	float64	True
Order ID	order_id	event	str	string	False
Item count	item_count	event	int	uint16	True
Revenue	revenue	event	float	float64	True
Revenue (Subtotal)	revenue_subtotal	event	float	float64	True
Revenue (Tax)	revenue_tax	event	float	float64	True
Revenue (Shipping)	revenue_shipping	event	float	float64	True
Revenue (Discount)	revenue_discount	event	float	float64	True
Time until DOM is ready	timing_dom_interactive	event	int	uint32	True
Time to interact	timing_event_end	event	int	uint32	True
Consent form view source	consent_source	event	[int, str]	uint8	True
Consent form interaction type	consent_form_button	event	[int, str]	uint8	True
Consent scope	consent_scope	event	[int, str]	uint8	True
Consent action	consent_action	event	[int, str]	uint8	True
Analytics consent	consent_type_analytics	event	int(0,1)	uint8	True
AB testing personalization consent	consent_type_ab_testing_personalization	event	int(0,1)	uint8	True
Conversion tracking consent	consent_type_conversion_tracking	event	int(0,1)	uint8	True
Marketing automation consent	consent_type_marketing_automation	event	int(0,1)	uint8	True
Remarketing consent	consent_type_remarketing	event	int(0,1)	uint8	True
User feedback consent	consent_type_user_feedback	event	int(0,1)	uint8	True
Custom consent 1	consent_type_custom_1	event	int(0,1)	uint8	True
Event custom dimension 1	event_custom_dimension_1	event	str	string	False
Event custom dimension 2	event_custom_dimension_2	event	str	string	False
Event custom dimension 3	event_custom_dimension_3	event	str	string	False
Event custom dimension 4	event_custom_dimension_4	event	str	string	False
Event custom dimension 5	event_custom_dimension_5	event	str	string	False
Event custom variable key 1	event_custom_variable_key_1	event	str	string	False
Event custom variable value 1	event_custom_variable_value_1	event	str	string	False
Event custom variable key 2	event_custom_variable_key_2	event	str	string	False
Event custom variable value 2	event_custom_variable_value_2	event	str	string	False
Event custom variable key 3	event_custom_variable_key_3	event	str	string	False
Event custom variable value 3	event_custom_variable_value_3	event	str	string	False
Event custom variable key 4	event_custom_variable_key_4	event	str	string	False
Event custom variable value 4	event_custom_variable_value_4	event	str	string	False
Event custom variable key 5	event_custom_variable_key_5	event	str	string	False
Event custom variable value 5	event_custom_variable_value_5	event	str	string	False
Session custom dimension 1	session_custom_dimension_1	session	str	string	False
Session custom dimension 2	session_custom_dimension_2	session	str	string	False
Session custom dimension 3	session_custom_dimension_3	session	str	string	False

Table 2 – continued from previous page

Dimension Name	Column ID	Scope	Type	Database Type	Nullab
Session custom dimension 4	session_custom_dimension_4	session	str	string	False
Session custom dimension 5	session_custom_dimension_5	session	str	string	False
Session custom variable key 1	session_custom_variable_key_1	session	str	string	False
Session custom variable value 1	session_custom_variable_value_1	session	str	string	False
Session custom variable key 2	session_custom_variable_key_2	session	str	string	False
Session custom variable value 2	session_custom_variable_value_2	session	str	string	False
Session custom variable key 3	session_custom_variable_key_3	session	str	string	False
Session custom variable value 3	session_custom_variable_value_3	session	str	string	False
Session custom variable key 4	session_custom_variable_key_4	session	str	string	False
Session custom variable value 4	session_custom_variable_value_4	session	str	string	False
Session custom variable key 5	session_custom_variable_key_5	session	str	string	False
Session custom variable value 5	session_custom_variable_value_5	session	str	string	False
Timestamp	timestamp	session	date	not applicable	False
Local hour	local_hour	session	int	not applicable	False
Time of redirections	redirections_time	event	int	not applicable	True
Domain Lookup Time	domain_lookup_time	event	int	not applicable	True
Server Connection Time	server_connection_time	event	int	not applicable	True
Server Response Time	server_response_time	event	int	not applicable	True
Page Rendering Time	page_rendering_time	event	int	not applicable	True
IPv4 address	ipv4_address	session	ipv4	not applicable	True
IPv6 address	ipv6_address	session	ipv6	not applicable	True
Website Name	website_name	session	[str, str]	not applicable	False

Note: Please note that the number of available custom slots (dimensions, variables) depends on your organisation's configuration.

1.1.3 Transformations

The tables below list all transformations that may be used to transform dimensions to metrics or different dimensions.

Table 3: Dimension To Metric Transformations

Transformation Name	Transformation ID	Source Types	Result Type
Unique Count	unique_count	int, str	int
Min	min	int, float	(as source)
Max	max	int, float	(as source)
Average	average	int, float	float
Median	median	int, float	(as source)
Sum	sum	int, float	(as source)

Table 4: Dimension To Dimension Transformations

Transformation Name	Transformation ID	Source Types	Result Type
Date To Day	to_date	datetime, date	date
Date To Start Of Hour	to_start_of_hour	datetime	datetime
Date To Start Of Week	to_start_of_week	datetime, date	date
Date To Start Of Month	to_start_of_month	datetime, date	date
Date To Start Of Quarter	to_start_of_quarter	datetime, date	date
Date To Start Of Year	to_start_of_year	datetime, date	date
Date To Hour Of Day	to_hour_of_day	datetime	int
Date To Day Of Week	to_day_of_week	datetime, date	int
Date To Month Number	to_month_number	datetime, date	int
URL To Path	to_path	str	str
URL To Domain	to_domain	str	str
URL Strip Query String	strip_qs	str	str

1.2 Integrations

Documents in this section describe the structure of data provided by third-party integrations.

1.2.1 Google Ads

The *HTTP API* supports querying Google Ads data just like the internal analytics data.

Note: You must configure the Google Ads integration before any data from it will become available. This can be done in the **Settings / Integrations** application's section.

Metrics

The table below lists metrics provided by Google Ads integration.

Table 5: Google Ads Metrics

Metric Name	Column ID	Scope	Type
Impressions (Google Ads)	google_ads_impressions	external	int
Clicks (Google Ads)	google_ads_clicks	external	int
Cost (Google Ads)	google_ads_cost	external	float
Average CPC (Google Ads)	google_ads_average_cpc	external	float
CTR (Google Ads)	google_ads_ctr	external	float
ROAS (Google Ads)	google_ads_roas	session	float

Dimensions

The table below lists dimensions provided by Google Ads integration.

Note: "Database type" column presents the type of source column of the dimension (in case of enum - type of the ID, in case of dynamic dimensions - not applicable).

Table 6: Google Ads Dimensions

Dimension Name	Column ID	Scope	Type	Database Type	Nullable	Notes
Source	source	session	str	string	False	
Medium	medium	session	str	string	False	
Source/Medium	source_medium	session	str	string	False	
Keyword	keyword	session	str	string	False	
Device type	device_type	session	[int, str]	uint8	True	`device_type.json`
Session entry URL	session_entry_url	session	str	string	False	
Timestamp	timestamp	session	date	not applicable	False	by default in Raw data API
Website Name	website_name	session	[str, str]	not applicable	False	website UUID
Customer ID (Google Ads)	google_ads_customer_id	session	str_id	string	False	
Customer Name (Google Ads)	google_ads_customer_name	session	[str, str]	not applicable	False	
Campaign ID (Google Ads)	google_ads_campaign_id	session	int_id	int64	False	
Campaign Name (Google Ads)	google_ads_campaign_name	session	[int, str]	not applicable	False	
Ad Group ID (Google Ads)	google_ads_ad_group_id	session	int_id	int64	False	
Ad Group Name (Google Ads)	google_ads_ad_group_name	session	[int, str]	not applicable	False	
Ad Group Ad ID (Google Ads)	google_ads_ad_group_ad_id	session	str_id	string	False	
Ad Group Ad Network Type (Google Ads)	google_ads_ad_network_type	session	[int, str]	string	False	`google_ads_ad_network_type.json`
Ad Group Keyword Match Type (Google Ads)	google_ads_keyword_match_type	external	[int, str]	string	False	`google_ads_keyword_match_type.json`, not available in Raw data API

Mixed Queries

It is possible to request both internal analytics and Google Ads metrics in a single query (for example: "Sessions" and "Clicks (Google Ads)"), however **only the common dimensions listed below** may be used in such queries.

Note: "Database type" column presents the type of source column of the dimension (in case of enum - type of the ID, in case of dynamic dimensions - not applicable).

Table 7: Common Dimensions

Dimension Name	Column ID	Scope	Type	Database Type	Nullable	Notes
Source	source	session	str	string	False	
Medium	medium	session	str	string	False	
Source/Medium	source_medium	session	str	string	False	
Keyword	keyword	session	str	string	False	
Device type	device_type	session	[int, str]	uint8	True	`device_type.json`
Session entry URL	session_entry_url	session	str	string	False	
Timestamp	timestamp	session	date	not applicable	False	by default in Raw data API
Website Name	website_name	session	[str, str]	not applicable	False	website UUID
Customer ID (Google Ads)	google_ads_customer_id	session	str_id	string	False	
Customer Name (Google Ads)	google_ads_customer_name	session	[str, str]	not applicable	False	
Campaign ID (Google Ads)	google_ads_campaign_id	session	int_id	int64	False	
Campaign Name (Google Ads)	google_ads_campaign_name	session	[int, str]	not applicable	False	
Ad Group ID (Google Ads)	google_ads_ad_group_id	session	int_id	int64	False	
Ad Group Name (Google Ads)	google_ads_ad_group_name	session	[int, str]	not applicable	False	
Ad Group Ad ID (Google Ads)	google_ads_ad_group_ad_id	session	str_id	string	False	
Ad Group Ad Network Type (Google Ads)	google_ads_ad_network_type	session	[str, str]	string	False	`google_ads_ad_network_type.json`

Warning: Using dimensions that are not explicitly listed in the table above in such queries (either as query columns or as filters) will result in a **Bad Request** response.

1.2.2 Google Search Console

The *HTTP API* supports querying Google Search Console data just like the internal analytics data.

Note: You must configure the Google Search Console integration before any data from it will become available. This can be done in the **Settings / Integrations** application's section.

Metrics

The table below lists metrics provided by Google Search Console integration.

Table 8: Google Search Console Metrics

Metric Name	Column ID	Scope	Type
Clicks (search engine)	search_engine_clicks	external	int
Impressions (search engine)	search_engine_impressions	external	int
CTR (search engine)	search_engine_ctr	external	float
Average position (search engine)	search_engine_average_position	external	float

Dimensions

The table below lists dimensions provided by Google Search Console integration.

Note: "Database type" column presents the type of source column of the dimension (in case of enum - type of the ID, in case of dynamic dimensions - not applicable).

Table 9: Google Search Console Dimensions

Dimension Name	Column ID	Scope	Type	Database Type	Nullable	Notes
Source	source	session	str	string	False	
Medium	medium	session	str	string	False	
Source/Medium	source_medium	session	str	string	False	
Channel	referrer_type	session	[int, str]	uint8	False	`referrer_type.json`
Referrer URL	referrer_url	session	str	string	False	
Device type	device_type	session	[int, str]	uint8	True	`device_type.json`
Continent	location_continent_iso_code	session	[str, str]	string(2)	True	`location_continent_iso_code.json`
Country	location_country_name	session	[str, str]	string	True	ISO 3166-2 codes (e.g. "PL")
Session entry URL	session_entry_url	session	str	string	False	
Timestamp	timestamp	session	date	not applicable	False	by default in Raw data API
Search engine keyword	search_engine_keyword	external	str	string	False	not available in Raw data API
Website Name	website_name	session	[str, str]	not applicable	False	website UUID

Mixed Queries

It is possible to request both internal analytics and Google Search Console metrics in a single query (for example: "Sessions" and "Clicks (search engine)"), however **only the common dimensions listed below** may be used in such

queries.

Note: "Database type" column presents the type of source column of the dimension (in case of enum - type of the ID, in case of dynamic dimensions - not applicable).

Table 10: Common Dimensions

Dimension Name	Column ID	Scope	Type	Database Type	Nullable	Notes
Source	source	session	str	string	False	
Medium	medium	session	str	string	False	
Source/Medium	source_medium	session	str	string	False	
Channel	referrer_type	session	[int, str]	uint8	False	`referrer_type.json`
Referrer URL	referrer_url	session	str	string	False	
Device type	device_type	session	[int, str]	uint8	True	`device_type.json`
Continent	location_continent_iso_code	session	[str, str]	string(2)	True	`location_continent_iso_code.json`
Country	location_country_name	session	[str, str]	string	True	ISO 3166-2 codes (e.g. "PL")
Session entry URL	session_entry_url	session	str	string	False	
Timestamp	timestamp	session	date	not applicable	False	by default in Raw data API
Website Name	website_name	session	[str, str]	not applicable	False	website UUID

Warning: Using dimensions that are not explicitly listed in the table above in such queries (either as query columns or as filters) will result in a **Bad Request** response.

1.2.3 SharePoint

Once SharePoint integration is enabled, additional metrics and dimensions will become available in the *HTTP API*.

Metrics

The table below lists metrics available with SharePoint integration.

Table 11: SharePoint Metrics

Metric Name	Column ID	Scope	Type
SharePoint shares	sharepoint_shares	session	int
SharePoint likes	sharepoint_likes	session	int
SharePoint comments	sharepoint_comments	session	int
SharePoint promotions	sharepoint_promotions	session	int
SharePoint creations	sharepoint_creations	session	int
SharePoint edits	sharepoint_edits	session	int
SharePoint deletions	sharepoint_deletions	session	int
SharePoint opens	sharepoint_opens	session	int
SharePoint uploads	sharepoint_uploads	session	int
SharePoint item views	sharepoint_item_views	session	int
SharePoint item attachment views	sharepoint_item_attachment_views	session	int
SharePoint item shares	sharepoint_item_shares	session	int

Dimensions

The table below lists dimensions available with SharePoint integration.

Note: “Database type” column presents the type of source column of the dimension (in case of enum - type of the ID, in case of dynamic dimensions - not applicable).

Table 12: SharePoint Dimensions

Dimension Name	Column ID	Scope	Type	Database Type	Nullable	Notes
SharePoint display name	sharepoint_display_name	session	str	string	True	
SharePoint office	sharepoint_office	session	str	string	True	
SharePoint department	sharepoint_department	session	str	string	True	
SharePoint job title	sharepoint_job_title	session	str	string	True	
SharePoint shares in session	session_total_sharepoint_shares	session	int	uint16	False	
SharePoint likes in session	session_total_sharepoint_likes	session	int	uint16	False	
SharePoint comments in session	session_total_sharepoint_comments	session	int	uint16	False	
SharePoint promotions in session	session_total_sharepoint_promotions	session	int	uint16	False	
SharePoint creations in session	session_total_sharepoint_creations	session	int	uint16	False	
SharePoint edits in session	session_total_sharepoint_edits	session	int	uint16	False	
SharePoint deletions in session	session_total_sharepoint_deletions	session	int	uint16	False	
SharePoint opens in session	session_total_sharepoint_opens	session	int	uint16	False	
SharePoint uploads in session	session_total_sharepoint_uploads	session	int	uint16	False	
SharePoint item views in session	session_total_sharepoint_item_views	session	int	uint16	False	
SharePoint item attachment views in session	session_total_sharepoint_item_attachment_views	session	int	uint16	False	
SharePoint item shares in session	session_total_sharepoint_item_shares	session	int	uint16	False	
SharePoint action	sharepoint_action	event	[int, str]	uint8	True	`sharepoint_action.json`
SharePoint object type	sharepoint_object_type	event	[int, str]	uint8	True	`sharepoint_object_type.json`
SharePoint content type	sharepoint_content_type	event	str	string	True	
SharePoint author	sharepoint_author	event	str	string	True	
SharePoint author's display name	sharepoint_author_display_name	event	str	string	True	
SharePoint author's office	sharepoint_author_office	event	str	string	True	
SharePoint author's department	sharepoint_author_department	event	str	string	True	
SharePoint author's job title	sharepoint_author_job_title	event	str	string	True	
SharePoint file url	sharepoint_file_url	event	str	string	True	
SharePoint file type	sharepoint_file_type	event	str	string	True	

1.3 HTTP API

1.4 Object management API

1.5 Metrics Mapping

Names of metrics used in API are different in Analytics classic and Analytics new. If you're migrating to the Analytics new API then below metrics mapping table will be helpful to you. You can find there a list of metrics used in Analytics classic and their corresponding names in the Analytics new.

Note: Event dimensions can only be used with metrics calculated for an event dimension.

1.5.1 Simple Metrics

Metric name	Legacy API	New API
Events	nb_actions	`{"column_id": "events"}`
Sessions	nb_visits	`{"column_id": "sessions"}`
Visitors	nb_uniq_visitors	`{"column_id": "visitors"}`
Users	nb_users	`{"column_id": "users"}`
Page views	nb_pageviews nb_hits	`{"column_id": "page_views"}`
Unique page views	nb_uniq_pageviews	`{"column_id": "unique_page_views"}`
Outlinks	nb_outlinks	`{"column_id": "outlinks"}`
Unique outlinks	nb_uniq_outlinks	`{"column_id": "unique_outlinks"}`
Downloads	nb_downloads	`{"column_id": "downloads"}`
Unique downloads	nb_uniq_downloads	`{"column_id": "unique_downloads"}`
Searches	-	`{"column_id": "searches"}`
Unique searches	nb_searches	`{"column_id": "unique_searches"}`
Custom events	nb_events	`{"column_id": "custom_events"}`
Unique custom events	-	`{"column_id": "unique_custom_events"}`
Content impressions	nb_impressions	`{"column_id": "content_impressions"}`
Unique content impressions	-	`{"column_id": "unique_content_impressions"}`
Content interactions	nb_interactions	`{"column_id": "content_interactions"}`

Continued on next page

Table 13 – continued from previous page

Metric name	Legacy API	New API
Unique content interactions	-	`{"column_id": "unique_content_interactions"}`
Content interaction rate	interaction_rate	`{"column_id": "content_interaction_rate"}`
Goal conversions	nb_conversions **Warning:** ecommerce conversion was reported as goal conversion for goal_id 0	`{"column_id": "goal_conversions"}`
Ecommerce conversions		`{"column_id": "ecommerce_conversions"}`
Goal conversions (specific goal)	goal_<idGoal>_nb_conversions	`{"column_id": "goal_conversions", "goal_id": 1}`
Ecommerce abandoned carts	-	`{"column_id": "ecommerce_abandoned_carts"}`
Goal conversion rate	conversion_rate	`{"column_id": "goal_conversion_rate"}`
Ecommerce conversion rate	-	`{"column_id": "ecommerce_conversion_rate"}`
Entries	entry_nb_visits	`{"column_id": "entries"}`
Entry rate	-	`{"column_id": "entry_rate"}`
Exits	exit_nb_visits	`{"column_id": "exits"}`
Exit rate	exit_rate **Warning:** definition switches depending on report	`{"column_id": "exit_rate"}`
Exit rate (events)		`{"column_id": "exit_rate_events"}`
Bounces	bounce_count	`{"column_id": "bounces"}`
Bounce rate	bounce_rate **Warning:** definition switches depending on report	`{"column_id": "bounce_rate"}`
Bounce rate (events)		`{"column_id": "bounce_rate_events"}`
% of returning visitors	-	`{"column_id": "returning_visitors_rate"}`
Visitor IPs	-	`{"column_id": "visitor_ips"}`

Continued on next page

Table 13 – continued from previous page

Metric name	Legacy API	New API
Events per session **Warning:** does not include all event types	nb_actions_per_visit	`{"column_id": "events_per_session"}`
Unique purchases	-	`{"column_id": "unique_purchases"}`

1.5.2 Calculated Metrics

Warning: This table does not include every single possible combination of a dimension and a transformation, just some common examples.

Metric name	Legacy API	New API
Sum of goal revenue	revenue ecommerce revenue was reported as goal revenue for goal_id 0	`{"column_id": "goal_revenue", "transformation_id": "sum"}`
Sum of ecommerce revenue		`{"column_id": "revenue", "transformation_id": "sum"}`
Sum of goal revenue (specific goal)	goal_<idGoal>_revenue	`{"column_id": "goal_revenue", "transformation_id": "sum", "goal_id": 1}`
Average generation time	avg_time_generation	`{"column_id": "page_generation_time", "transformation_id": "average"}`
Max generation time	max_time_generation	`{"column_id": "page_generation_time", "transformation_id": "max"}`
Average time on page	avg_time_on_page	`{"column_id": "time_on_page", "transformation_id": "average"}`
Sum of time on page	sum_time_spent	`{"column_id": "time_on_page", "transformation_id": "sum"}`
Sum of session time	sum_visit_length	`{"column_id": "session_total_time", "transformation_id": "sum"}`
Average session time	avg_time_on_site	`{"column_id": "session_total_time", "transformation_id": "average"}`
Max events in session	max_actions	`{"column_id": "session_total_events", "transformation_id": "max"}`
Sum of custom events value	sum_event_value	`{"column_id": "custom_event_value", "transformation_id": "sum"}`
Average custom events value	avg_event_value	`{"column_id": "custom_event_value", "transformation_id": "average"}`

1.5.3 Not available

Name	Legacy API	Closest equivalent in Analytics new
Number of sessions that converted a goal	nb_visits_converted	Sessions metric with filter goal_conversions > 0
Number of custom events which had a value set	nb_events_with_value	Custom events metric with filter custom event value > 0
Number of hits that included generation time information	nb_hits_with_time_generation	Page views metric with filter page_generation_time > 0
Number of unique visitors that started their visit on this page	entry_nb_uniq_visitors	-
Number of page views for sessions that started on this page	entry_nb_actions	Entries metric (all entries are page views now)
Time spent, in seconds, by sessions that started on this page	entry_sum_visit_length	-
Number of sessions that started on this page, and bounced	entry_bounce_count	Bounces metric
Number of unique visitors that ended their visit on this page	exit_nb_uniq_visitors	-
Sum of daily unique visitors over days in the period	sum_daily_nb_uniq_visitors	No longer relevant, unique visitors are calculated across any period
Sum of daily unique visitors that started their visit on this page	sum_daily_entry_nb_uniq_visitors sum_daily_exit_nb_uniq_visitors	
Number of times this action was done after a site search	nb_hits_following_search	-

CHAPTER 2

Data Collection

2.1 Web

2.1.1 Guides

Installing Tracking code

Using Tag Manager's snippet is the recommended and also the easiest way of installing tracking code on your website. When Tag Manager is added to the site, it automatically starts tracking actions using "Piwik PRO Analytics template".

If you do not have Tag Manager on your website yet, follow Install a container article to install it. In short, you will need to:

1. Sign in to your PPAS with your admin or Super User account.
2. Click on the menu button on the top left.
3. Click on the "Websites" position.
4. Choose the website for which you want to implement a tracking code.
5. Select the "Installation" tab.
6. The Tag Manager's snippet for your website is displayed under the "Website code for asynchronous tags" or "Website code for synchronous tags".

In case you do not want to install Tag Manager on your website, you can install tracking code via JavaScript Tracking Snippet. Guide how to do it is available here: *Installing tracking code via code snippet*.

Page views

Page view is the most basic type of the tracked event. It represents a single page viewing action. By default it's triggered only once as soon as the HTML content is loaded to the browser with the *trackPageView* function.

```
_paq.push(["trackPageView"]);
```

Note: It's not required for the session to start with the page view or even involve them in any other way.

Note: We recommend to trigger this function more than once for Singe Page Applications (SPA). That way you'll create additional "virtual" page view as the visitor travels across your app.

User ID

User ID is an additional parameter that allows you to aggregate data. When set you will be able to search through sessions by this parameter, filter reports through it or create Multi attribution reports using *User ID*. You can learn more about User ID here. To set up *User ID* with your JavaScript Tracking snippet follow this guide.

To set up *User ID* parameter add a call to *setUserId* to your tracking code:

```
_paq.push(["setUserId", "user-name@example.com"]);
```

Note:

1. Invoking `setUserId` won't send any tracking request. To add *User ID* to tracked data, you have to call `setUserId` before *trackPageView*. It does not have to be the first one. Another way is to send ping request with *ping*, after setting the *User ID*.
2. *User ID* can't be longer than 1024 bytes. It will be 1024 characters if you use only ASCII characters, but Unicode characters may require more bytes per character, so you should make sure that Unicode identifier cut down to 1024 bytes is still unique.
3. *User ID* should be a unique value for each user. Otherwise metrics for different users might be merged in the reports.
4. Usually a *User ID* value is an user email, because this is the identifier that users use to log in to a website and it fulfils above requirements.

It is a good practice to remove value of *User ID*, when the user logs out. Otherwise *User ID* value might affect session of other users, if they share the same device. To remove *User ID* value call *resetUserId*.

```
_paq.push(["resetUserId"]);
```

Full abstract example, might look like this:

```
var user = getUserData();
if (user.isLogged) {
  _paq.push(["setUserId", user.login]);
} else {
  _paq.push(["resetUserId"]);
}
```

Warning: Do not clear *User ID* by setting it to some seemingly empty value, like `_paq.push(["setUserId", " "]);` or `_paq.push(["setUserId", ""]);`. This way some value might

be still send to Collecting & Processing Pipeline. What seems to be an empty value to a human, might not be to a machine. Only using `resetUserId` will properly clear the *User ID* value.

Note: Use of `resetUserId` is necessary only when clicking on log out button does not result in a page reload. For example, when your page is a Single Page Application, or user logout is initiated by a widget and the widget does not cause the webpage to reload, then you have to call `resetUserId`. Otherwise, when page reloads on logout, then a call to `resetUserId` is not a necessity, but sill, a good practice.

Note: Set up a user ID article shows an easy way to get *User ID* without modifying the source code of your website.

Custom Events

Custom events enable tracking visitor actions that are not predefined in the existing JavaScript Tracking Client API, allowing web analysts to accurately measure and analyze any domain. Many integrations, including those offered by Tag Manager, use custom events for tracking actions detectable only on client-side, e.g. scrolling a page, interacting with a video player, filling forms, etc.

A custom event consists of the following properties:

- **category** - Describes the category of an event, e.g. *video*, *form*, *scroll*
- **action** - Describes what action happened on a website, e.g. *video-play*, *video-pause*, *form-focus*, *scroll-progress*
- **name** (optional) - Usually contains the name of an action target, e.g. the name of a video, label of a form field, name of the scrolled article
- **value** (optional) - Additional numeric value carried with an event, e.g. number of seconds a video has been watched for, how far (in percentages) an article has been scrolled

Warning: Consider designing categories and actions upfront and documenting them at start and as they change. Follow one naming convention, e.g. *snake_case*, *kebab-case*, *camelCase*. This will minimize the risk of making mistakes and having to debug the tracking implementation.

Tracking a custom event together with a page view is straightforward - simply call *trackEvent* function after the page view.

```
_paq.push(["trackPageView"]);
_paq.push(["trackEvent", "assignment", "assignment-submitted", "Math - Trigonometry -␣
↪assignment 4", 10]);
```

The snippet above tracks a custom event with category *assignment*, action *assignment-submitted*, name *Math - Trigonometry - assignment 4* and value *10* (which might indicate the number of pages in a submitted document).

Custom event name and custom event value are optional. You can skip them if they are not meaningful in your use case.

```
_paq.push(["trackEvent", "category", "action"]); // skip both name and value
_paq.push(["trackEvent", "category", "action", "name"]); // skip only value
_paq.push(["trackEvent", "category", "action", undefined, 10.0]); // skip only name
```

Often we want to track events triggered by visitor's actions, sometime after the page has loaded. One way to do that is to add tracking code to event handling attributes of HTML elements, e.g. `onclick` attribute of `button` element.

```
<button onclick="likePost(); _paq.push(['trackEvent', 'social', 'like-post', 'top-10-
↪attractions-in-london'])">Like</button>
```

Warning: When tracking custom events this way, make sure HTML events trigger both the intended action and tracking code.

Note: Notice the change in string quotation style. Because `onclick` attribute content is quoted with double quotes, to avoid conflicts, strings in `_paq.push` have been surrounded with single quotes.

Tracking more sophisticated events might require attaching listeners to the DOM elements in a script and using *trackEvent* inside, for example:

```
<script>
    var maxScroll = 0.0;
    window.addEventListener("scroll", function (event) {
        var currentScroll = calculateScrollBetween0And1(event);
        if (currentScroll >= maxScroll + 0.1) {
            _paq.push(["trackEvent", "scroll", "page-scroll", document.title,␣
↪currentScroll]);
            maxScroll = currentScroll;
        }
    });
</script>
```

Note: Analytics for advanced analysts is a series of guides explaining how to track many different actions with custom events in Tag Manager. Check it out if you're looking for some inspiration!

Site search

Site search tracking gives you insight into how visitors interact with the search engine on your website - what they search for and how many results they get back.

Our data collecting and processing pipeline automatically converts page views into site search events if the URL contains site search query parameters: `q`, `query`, `s`, `search`, `searchword` and `keyword`. You can customize these parameters on the website settings page. Site search events can also be tracked manually by calling *trackSiteSearch* function. It allows specifying not only the keyword and category, but also the number of results and additional custom dimensions.

trackSiteSearch accepts the following parameters:

- **keyword** - what term someone looked for
- **category** (optional) - which category the search was in
- **results** (optional) - how many search results were returned
- **dimensions** (optional) - custom dimensions to send along the site search

It is used like this:

```
_paq.push(["trackSiteSearch", "les paul", "electric guitars", 5, { dimension10: "amber
↪" }]);
```

In this case, we track site search with keyword *les paul*, category *electric guitars*, *5* search results and custom dimension *10* with value *amber*.

The optional parameters might be skipped or replaced with `undefined` to indicate no value.

```
_paq.push(["trackSiteSearch", "playstation"]); // only keyword provided
_paq.push(["trackSiteSearch", "playstation", "consoles"]); // only keyword and␣
↪category provided
_paq.push(["trackSiteSearch", "playstation", undefined, 5]); // only keyword and␣
↪results count provided
```

Warning: If you can't or don't want to rely on automatic site search detection from URL parameters, call `trackSiteSearch` function instead of `trackPageView` on the search results page. Using both methods might result in a duplication of site search events.

E-commerce

JavaScript API supports 3 types of e-commerce interactions: *Category and product views*, *Cart updates* and *Orders*.

Tracking category and product views

Usually, the first e-commerce-related action a visitor performs on a website is browsing products. *setEcommerceView* function allows us to track both category views and product views.

To track a category view, use *setEcommerceView* function **before** tracking the page view, like this:

```
// set category to "Smartphones"
_paq.push(["setEcommerceView", undefined, undefined, "Smartphones"]);

// track page view
_paq.push(["trackPageView"]);
```

The same function can be used for tracking product views. Again, it must be called **before** tracking a page view. Example:

```
// set product with...
_paq.push(["setEcommerceView",
    "71253029",              // SKU (stock-keeping unit)
    "SUPER Phone A40 White", // name
    "Smartphones",           // category
    1499.99                  // price
]);

// track page view
_paq.push(["trackPageView"]);
```

`category` parameter of the *setEcommerceView* function accepts not only string values, but also arrays of strings. This is useful for tracking products that belong to more than one category, or tracking pages that list products from multiple categories.

```
// set product with...
_paq.push(["setEcommerceView",
    "00492710",                   // SKU (stock-keeping unit)
    "SUPER Watch B20 Silver",     // name
    ["New offer", "Smartwatches"], // categories
    700.00                        // price
]);

// track page view
_paq.push(["trackPageView"]);
```

Tracking cart updates

Another type of e-commerce activity you can track is an update of a shopping cart. With it, we are able to measure how often visitors don't complete the ordering process and what products stay in abandoned carts.

Tracking a cart update has two steps: registering items from the cart and sending them. The following example uses two functions - *addEcommerceItem* and *trackEcommerceCartUpdate* - to achieve exactly that.

```
// visitor added one chocolate bar to an empty shopping cart

// register chocolate bar with...
_paq.push(["addEcommerceItem",
    "82775027",                 // SKU (stock-keeping unit)
    "MEGA Milk Chocolate 200g", // name
    "Candy",                    // category
    6.00,                       // price
    1                           // quantity
]);

// track cart update with a total value of 6.00
_paq.push(["trackEcommerceCartUpdate", 6.00]);
```

This code snippet sends a cart update event with a cart containing one item (SKU *candy-12837*, name *MEGA Milk Chocolate 200g*, category *Candy*, price *6.00*) and having total value of *6.00*.

The list of registered items is stored only in memory. **Reloading the page will clear the list** and the previously registered items will have to be added again.

```
// visitor added one mango fruit to a shopping cart with one chocolate bar

// register previously added items
_paq.push(["addEcommerceItem", "82775027", "MEGA Milk Chocolate 200g", "Candy", 6.00,␣
↪1]);

// register the new item
_paq.push(["addEcommerceItem", "01809926", "FRUTASTIC Mango", "Fruits & vegetables",␣
↪4.00, 1]);

// track cart update with a total value of 10.00
_paq.push(["trackEcommerceCartUpdate", 10.00]);
```

Note: If you are not sure what items have been registered, use *getEcommerceCart* function.

```
_paq.push([function() { console.log(this.getEcommerceItems()); }]);
```

Because single page applications do not refresh the page when a visitor manipulates the cart, an e-commerce implementation in SPAs must either:

1. Clear the cart using *clearEcommerceCart* and register all items from the cart before tracking cart update, e.g.

```
// visitor added one chocolate bar to an empty shopping cart
_paq.push(["clearEcommerceCart"]);
_paq.push(["addEcommerceItem", "82775027", "MEGA Milk Chocolate 200g", "Candy", 6.00,␣
↪1]);
_paq.push(["trackEcommerceCartUpdate", 6.00]);

// visitor added one mango fruit to a shopping cart with one chocolate bar
_paq.push(["clearEcommerceCart"]);
_paq.push(["addEcommerceItem", "82775027", "MEGA Milk Chocolate 200g", "Candy", 6.00,␣
↪1]);
_paq.push(["addEcommerceItem", "01809926", "FRUTASTIC Mango", "Fruits & vegetables",␣
↪4.00, 1]);
_paq.push(["trackEcommerceCartUpdate", 10.00]);

// visitor removed one chocolate from a shopping cart with one chocolate bar and one␣
↪mango
_paq.push(["clearEcommerceCart"]);
_paq.push(["addEcommerceItem", "01809926", "FRUTASTIC Mango", "Fruits & vegetables",␣
↪4.00, 1]);
_paq.push(["trackEcommerceCartUpdate", 4.00]);
```

2. Replicate visitor's interactions with the cart using functions *addEcommerceItem*, *removeEcommerceItem*, *clearEcommerceCart*.

```
// visitor added one chocolate bar to an empty shopping cart
_paq.push(["addEcommerceItem", "82775027", "MEGA Milk Chocolate 200g", "Candy", 6.00,␣
↪1]);
_paq.push(["trackEcommerceCartUpdate", 6.00]);

// visitor added one mango fruit to a shopping cart with one chocolate bar
_paq.push(["addEcommerceItem", "01809926", "FRUTASTIC Mango", "Fruits & vegetables",␣
↪4.00, 1]);
_paq.push(["trackEcommerceCartUpdate", 10.00]);

// visitor removed one chocolate bar from a shopping cart with one chocolate bar and␣
↪one mango
_paq.push(["removeEcommerceItem", "82775027"]);
_paq.push(["trackEcommerceCartUpdate", 4.00]);
```

Tracking orders

Perhaps the most important element of an e-commerce implementation is tracking orders. Just like with *cart updates*, tracking orders has two steps: registering items that have been purchased and tracking the order. Registering items looks exactly the same - we use *addEcommerceItem*, *removeEcommerceItem* and *clearEcommerceCart*. The actual tracking of an order is done with a call to *trackEcommerceOrder* function.

```
// register all purchased items

_paq.push(["addEcommerceItem",
    "66251929",               // SKU
    "Red Unicorn Coffee Mug", // name
    "Tableware",              // category
    8.00,                     // price
    1                         // quantity
]);

_paq.push(["addEcommerceItem",
    "08273511",               // SKU
    "SUPER Blue Ink Pen 0.2", // name
    "Office products",        // category
    2.00,                     // price
    2                         // quantity
]);

// track order
_paq.push(["trackEcommerceOrder",
    "online-5289",            // ID
    16.00,                    // grand total (value + tax + discount + shipping)
    10.00,                    // sub total (value + tax + discount)
    1.00,                     // tax
    6.00,                     // shipping
    2.00                      // discount
]);
```

Warning: *trackEcommerceOrder* function clears the list with registered e-commerce items.

Content tracking

What is content tracking

Let's talk about a scenario in which simple page view tracking is not enough. It will just tell you which page was loaded, but it won't point out how visitors interact with the content on that particular page. Content impression and content interaction tracking feature fills that gap.

Content impression allows you to track what content is visible to the visitor. On the bigger pages it may tell what particular parts/blocks of it the visitor has reached. When they keep scrolling and new content is presented on the screen it will be tracked automatically. This is useful for ads and banners, but may be also attached to a image carousel or other forms of image galleries.

Now we know what block became visible on the screen, but we would also like to know how the visitor interacted with them. Content interaction tracking completes this feature. After particular block became visible on the viewport JavaScript Tracking Client will automatically record visitor clicks related to it.

JavaScript Tracking Client distinguishes three parts of the content structure: *content name*, *content piece* and *content target*. All together they are called *content block*.

- *Content name* - this is the title describing the content block, tracked data will be visible as an entry in the reports under that name
- *Content piece* - gives us the specific piece that was reached on the page (typically an image or other media)

- *Content target* - if the content block you want to track is an anchor, content target will contain the url this anchor links to

Enabling automatic content tracking

To enable automatic content tracking, call one of the following tracking functions:

- *trackAllContentImpressions* - tracks all content blocks present on page (visible and not visible)
- *trackVisibleContentImpressions* - continuously scans the window for visible blocks and sends an update if a new block shows up on screen

For more details visit the *Content tracking* section of the JavaScript Tracking Client API documentation.

Note: Automatic content tracking can be enabled in Tag Manager, as shown in Set up content tracking article.

But how does JavaScript Tracking Client know what blocks you would like to track?

There are two ways of marking HTML elements as content blocks: you must either add a special attribute `data-track-content` or class `piwikTrackContent`. Example:

```
<a href="http://example.com/image/abc.png" title="abc" data-track-content>
    first content block
</a>
<a href="http://example.com/image/def.png" title="def" class="piwikTrackContent">
    second content block
</a>
```

Content properties will be taken from HTML attributes of the content block element or any of its descendants:

- name comes from `data-content-name` attribute
- piece comes from `data-content-piece` attribute
- target comes from `data-content-target` attribute

If any of these attributes is missing, JavaScript Tracking Client will try extracting the value from other sources, using the following logic:

- piece will be taken from `src` attribute of an element with `piwikContentPiece` class or block element
- target will be taken from `href` attribute of an element with `piwikContentTarget` class, block element or piece element
- name will try to use piece value if present, otherwise it'll be taken from `title` attribute of block element, piece element or target element

However, these sources are sometimes unreliable and we recommend providing name, piece and target values in dedicated HTML attributes.

Note: `src` attribute is read when extracting content piece from common media elements: `img`, `embed`, `video`, `audio`. Other elements, like `object`, use more complex extraction logic.

Manual content tracking

If for some reason automatic content tracking does not suit your needs, you may still trigger *trackContentImpression* and *trackContentInteraction* JavaScript Tracking Client functions manually.

Example:

```
_paq.push(["trackContentImpression", "Ads", "Partner banner", "http://some-company.tld"]);

some_dom_node.addEventListener("click", function () {
    _paq.push(["trackContentInteraction", "bannerClicked", "Ads", "Partner banner", "http://some-company.tld"]);
});
```

Custom interaction tracking

There is also a third way to track content in more complicated situations. Automatic scenario will track clicks as a visitor interaction, but sometimes other activity may interest you more (e.g. hovering the mouse over a submit button of a form). In such scenarios you would like to enable automatic content impression tracking but trigger interaction tracking manually. Function *trackContentInteractionNode* lets you do that without the need to provide content name, piece and target in the call (it generates those values in the same way as the automatic method).

Example:

```
some_image_node.addEventListener("hover", function () {
    _paq.push(["trackContentInteractionNode", this, "submit-hover"]);
});
```

Note: It may be important that your "custom" interaction tracking is not later on doubled by the automatic one. To disable automatic content interaction tracking you should either apply `piwikContentIgnoreInteraction` CSS class or `data-content-ignoreinteraction` HTML attribute to the given element.

Examples

Simple HTML content block may look like this:

```
<a href="http://some-company.tld" title="Our business partner ad" data-track-content>
    Click here to see the website
</a>

// content name   = Our business partner ad
// content piece  = Unknown
// content target = http://some-company.tld
```

More advanced HTML content block with all attributes prepared (leaving nothing to chance) may look like this:

```
<a href="http://some-company.tld" title="Click here" data-track-content data-content-name="Our business partner ad">
    <img src="/images/business-partners/banners/some-company.png" data-content-piece />
</a>
```

(continues on next page)

(continued from previous page)

```

// content name   = Our business partner ad
// content piece  = /images/business-partners/banners/some-company.png
// content target = http://some-company.tld
```

Form submission:

```
<form data-track-content data-content-name="Survey form">
    <input type="submit" data-content-target="http://our-company.tld/form-handler" />
</form>

// content name   = Survey form
// content piece  = Unknown
// content target = http://our-company.tld/form-handler
```

Downloads and Outlinks

Download and outlinks are links on your site that point to content that normally can't be tracked (e.g. non-HTML files - downloads or pages outside your domain - outlinks). JavaScript Tracking Client allows you to track clicks on such links to let you know how popular they are.

Note: If you have modified default JS snippet provided by Tag Manager and still want to track download and/or outlinks, make sure that *enableLinkTracking* is called. It is enabled in default snippet, but if you use a custom one, then you have to enable it by yourself.

```
// Enable Download & Outlink tracking
_paq.push(["enableLinkTracking"]);
```

Downloads

Download data helps you learn which files are most popular on your site — be it a white paper, a case study, or a guide in PDF. Piwik PRO will automatically track clicks on such links as *Downloads*, and reports them in *Downloads* report.

JavaScript Tracking Client will automatically recognize download link by checking its target file extension.

Note: These are default file extensions indicating a download file: 7z, aac, apk, arc, arj, asf, asx, avi, azw3, bin, bz, bz2, csv, deb, dmg, doc, docx, epub, exe, flv, gif, gz, gzip, hqx, ibooks, jar, jpg, jpeg, js, mp2, mp3, mp4, mpg, mpeg, mobi, mov, movie, msi, msp, odb, odf, odg, ods, odt, ogg, ogv, pdf, phps, png, ppt, pptx, qt, qtm, ra, ram, rar, rpm, sea, sit, tar, tbz, tbz2, tgz, torrent, txt, wav, wma, wmv, wpd, xls, xlsx, xml, z, zip

---

Examples of download link URL:

- file extension is at the very end of path (eg. `http://example.com/file.7z` or `http://example.com/article/file.7z?source=user#how-to`)
- file extension is at the end of query param value (eg. `http://example.com/article?click=file.7z&page=3` or `http://example.com/article?target=file.7z#how-to`)

Customizing list of file extensions

You can customize list of file extensions you want to track as downloads. For example, if you want to track only images as downloads, you can use setDownloadExtension function to replace the list like this:

```
// track clicks on images links (eg. <a href="image.png">) only
_paq.push(["setDownloadExtensions", "png|jpg|webp|gif"]);
```

You can add new extensions, to an existing list with `addDownloadExtensions`:

```
// add other image formats
_paq.push(["addDownloadExtensions", "svg|xcf"]);
```

Or remove some of extensions from the existing list with `removeDownloadExtensions`:

```
_paq.push(["removeDownloadExtensions", "jpg|jpeg"]);
```

Manually marking links as downloads

If your download link can't be detected by extension, you still can tell JavaScript Tracking Client that link should be tracked as a download.

You can add a `download` attribute to a link HTML tag. eg.

```
<a href="/target-file" download>
```

Or if you have to be strict with your HTML, you can add a HTML tag class. Default classes are `piwik_download` and `piwik-download`. Eg.

```
<a href="/taget-file" class="piwik-download">
```

Additionally you can define your custom CSS classes for download links with our *JavaScript Tracking Client API*. Eg.

```
_paq.push(["setDownloadClasses", "custom-download-class"]);
_paq.push(["trackPageView"]);
```

or you can define a list of classes at once, by passing an array list of CSS classes:

```
_paq.push(["setDownloadClasses", ["custom-download-class", "other-download-class",
↪"another-class"]]);
_paq.push(["trackPageView"]);
```

and use that class in HTML code:

```
<a href="/taget-file" class="custom-download-class">
```

Note: You have to remember that using `setDownloadClasses` always overwrite current list of CSS classes.

Tracking downloads with inline Javascript

There is another alternative for above methods. You can track a download with inline JavaScript code. Insert inline code to HTML tag with `onclick` attribute:

```
<a href="https://piwik.pro/document-url" target="_blank" onClick="_paq.push([
↪'trackLink', 'https://piwik.pro/document-url', 'download']);">Download document</a>
```

Tracking downloads when using log importer

When you use the Log Importer, files with one of the file extensions listed above will be automatically tracked as downloads in Piwik PRO.

Outlinks

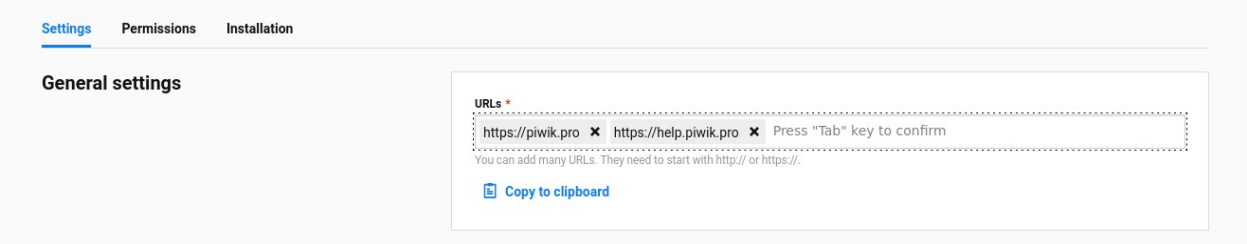
The Piwik PRO *Outlinks* report shows the list of external URLs that were clicked by your visitors. Outlinks are links that have different domain than those configured for your website. For example, if your visitors click on a link to *piwik.pro* and your website domain is *example.org*, this will be reported as an outlink, no matter if the website opens in current tab/window or a new one.

```
<a href="https://piwik.pro">Piwik PRO</a>
```

Configuring which domains are outlinks

When, for example, your main page is *piwik.pro* and you want to track views of *help.piwik.pro* without additional outlink click, you have to confgure JavaScript Tracking Client to recognize this additional domain. You can do it in two ways.

If you use default snippet provided by Tag Manager, you can configure it in website settings section of the Administration panel. Go to the Administration > Websites & apps > Settings > General settings > URLs. Add all the domains that should not be treated as outlinks.



If you don't use default snippet, you can use *setDomains* function of JavaScript Tracking Client API to set it.

```
_paq.push(["setDomains", ["help.piwik.pro", "piwik.pro", "*.other-domain.pro"]]);
_paq.push(["trackPageView"]);
```

Note: Each use of `setDomains` will overwrite previous configuration. If you use default snippet, it's safest to use Administration panel to set site domains and avoid using `setDomains` in custom tags to avoid race conditions.

Marking links as outlinks in HTML code

Similar to downloads, links can be set to be marked as outlinks manually, but only with CSS classes, you cannot use a HTML attribute to do that.

You can use one of default CSS classes: `piwik_link` or `piwik-link`. eg.

```
<a href="https://piwik.pro" class="piwik-link">Piwik PRO</a>
```

Or you can define your custom CSS classes for outlinks with *JavaScript Tracking Client API*.

```
// now all clicks on links with the css class "custom-link-class" will be counted as␣
↪outlinks
// you can also pass an array of strings
_paq.push(["setLinkClasses", "custom-link-class"]);
_paq.push(["trackPageView"]);
```

or a list of classes

```
_paq.push(["setLinkClasses", ["custom-link-class", "other-link-class"]]);
_paq.push(["trackPageView"]);
```

and using that class in HTML code

```
<a href="https://piwik.pro" class="custom-link-class">Piwik PRO</a>
```

Tracking outlinks with inline Javascript

Alternatively you can use an inline JavaScript code and `onclick` attribute to track any link as an outlink.

```
<a href="mailto:support@piwik.pro" target="_blank" onClick="_paq.push(['trackLink',
↪'https://piwik.pro/support-contact-form', 'link']);">Write us a message.</a>
```

Other link tracking options

Changing delay for link tracking

All link tracking introduces a slight delay between link click and click execution, so the browser won't exit the page before a click is tracked. The default value of such delay is 500ms, but you can modify it as you wish. You have to remember that if you set this value too low, it might be not enough to track the click, and if you set it too high, it will become noticable to viewer or the browser might ignore the delay entirely.

```
_paq.push(["setLinkTrackingTimer", 300]); // 300 milliseconds
_paq.push(["trackPageView"]);
```

Note: Link tracking will try to use more reliable navigator.sendBeacon method to send tracking requests on modern browsers, but legacy browsers that don't support this API will rely on page exit delay.

Disable download and outlink tracking

To explicitly disable link tracking you can use `disableLinkTracking` function. After adding it to tracking code, link clicks won't be tracked.

```
_paq.push(["disableLinkTracking"]);
```

Disabling link tracking with CSS classes

You can mark links that you do not with to track with CSS classes. JavaScript Tracking Client will ignore such links and won't track them.

```
_paq.push(["setIgnoreClasses", "do-not-track"]);
_paq.push(["trackPageView"]);
```

or a list of classes:

```
_paq.push(["setIgnoreClasses", ["dont-track-this", "this-either", "nor-this"]]);
_paq.push(["trackPageView"]);
```

and using that class in HTML code:

```
<a href="https://piwik.pro/document.pdf" class="dont-track-this">A document, that␣
↪should not be tracked.</a>
```

Tracking link clicks on pages with dynamically generated content

When you want to track clicks on the links, which are dynamically added to the HTML document, you have to call *enableLinkTracking* every time when the new links are added to the document.

For fully static pages calling *enableLinkTracking* once is enough, because each call adds listeners only for those links, which are currently present in the HTML document. So if you add new links to the document and you want to track them, you have to call *enableLinkTracking* each time that happens.

```
// Add click listeners to new links
_paq.push(["enableLinkTracking"]);
```

Note: You don't have to call *enableLinkTracking* if you are tracking links with inline JavaScript (with *trackLink*).

Goal tracking

At this point we have tracked many different types of events. We have regular page views, downloads, outlinks, custom events and others. Above them all there's one more event type we can track: a conversion. And goal tracking is about tracking conversions. If you can point out parts of your website/application more important from your business perspective, you could define those parts as goals. Visiting a specific landing page, submitting a contact form, downloading a PDF file with your product manual - these are popular examples of goal definitions. You can even define a goal based on the custom event you are tracking.

If a goal with automatic tracking is defined in Analytics, every time an events matching the goal's definition is tracked, we create an additional conversion event and save it along the original event. We call this procedure an "automatic conversion".

Note: Set up goals article shows how to define a goal triggered by visiting a specific page.

Alternatively, you can trigger a goal manually with the use of *trackGoal* function

```
// force conversion of the goal with ID 17
_paq.push(["trackGoal", 17]);
```

We call this procedure a "manual conversion". Manual conversion will send a standalone conversion event immediately and is not tied to any other event sent by tracker like automatic conversions.

Anonymous tracking

You can set JavaScript Tracking Client to mark requests to be anonymized. This feature can be useful when you want to use a consent manager on your website and collect full data only from those visitors who gave consent to be tracked.

To set JavaScript Tracking client to mark requests as anonymized call *setUserIsAnonymous*

```
_paq.push(["setUserIsAnonymous"]);
```

From now on all following requests sent by *trackPageView* or any other function that sends requests to *Collecting & Processing Pipeline*, will be marked as a request that should be anonymized. Learn more how Piwik PRO anonymizes visitors data.

Note: If your webpage reloads with each action performed by a visitor, eg. when visitor clicks a link or submits a form, then you have to call `setUserIsAnonymous` before first `trackPageView` on each page load. By default, JavaScript Tracking Client does not mark requests as anonymous.

When a visitor gives consent for tracking or you want to enrich anonymous data that is already sent for current visitor, call *deanonymizeUser*

```
_paq.push(["deanonymizeUser"]);
```

This will send special deanonymization request to *Collecting & Processing Pipeline*, that will enrich visitor's data with all the information that was stripped from previous requests.

To sum up:

1. You have to set JavaScript Tracking Client to anonymous mode with calling *setUserIsAnonymous*, at very start of your tracking code for all visitors, that you want to track anonymously (e.g. visitors that did not gave consent for tracking)
2. Prevent the call of *setUserIsAnonymous* for all of visitors that should not be anonymized (e.g. visitors that already gave consent)
3. To enrich already collected anonymous data of a visitor, you have to add a handler that will call *deanonymizeUser* when you want to denonymize the visitor (e.g. visitor clicked on a button to agree on tracking)

2.1.2 JavaScript Tracking Client

Installation

Installing tracking code via code snippet

Installation via snippet should only be carried out if the Tag Manager is not available or when options of "Piwik PRO Analytics template" do not let you configure your use case.

Note: We highly recommend using the template from the Tag Manager to set up tracking for the Analytics module (including customizations).

Note: Basic configuration will setup a single domain configuration. For other options, see: *Alternative configurations*.

This code should be added in the head section of the page just before the closing `</head>` tag. Additionally, the snippet must be configured in the following way:

- String `XXX-XXX-XXX-XXX-XXX` should be replaced with *app ID* (e.g. `efcd98a5-335b-48b0-ab17-bf43f1c542be`).
- String `https://your-instance-name.piwik.pro/` should be replaced with your PPAS instance address.

```
<!-- Piwik -->
<script type="text/javascript">
  var _paq = _paq || [];
  _paq.push(["trackPageView"]);
  _paq.push(["enableLinkTracking"]);
  (function() {
    var u="https://your-instance-name.piwik.pro/";
    _paq.push(["setTrackerUrl", u+"ppms.php"]);
    _paq.push(["setSiteId", "XXX-XXX-XXX-XXX-XXX"]);
    var d=document, g=d.createElement("script"), s=d.getElementsByTagName("script
↪")[0];
    g.type="text/javascript"; g.async=true; g.defer=true; g.src=u+"ppms.js"; s.
↪parentNode.insertBefore(g,s);
  })();
</script>
```

This code initializes the JavaScript Tracking Client in following ways:

1. Initializes the global `_paq` command queue that schedules commands to be run when the JavaScript Tracking Client library is loaded.
2. Schedules basic configuration of JavaScript Tracking Client using `_paq.push`.
3. Creates a `<script>` tag that asynchronously loads the JavaScript Tracking Client library.

When loading, the snippet is added on the page. The JavaScript Tracking Client will start tracking *visitor* actions starting with page view.

Alternative configurations

Tracking domains and all subdomains

To track all data between domain and all its subdomains, we must use cookies configured with the following snippet:

```
_paq.push(["setTrackerUrl", u+"ppms.php"]);
_paq.push(["setSiteId", "XXX-XXX-XXX-XXX-XXX"]);

// Share the tracking cookie across example.com, www.example.com, subdomain.example.
↪com, ...
_paq.push(["setCookieDomain", "*.example.com"]);

// Tell Piwik the website domain so that clicks on these domains are not tracked as
↪"Outlinks"
_paq.push(["setDomains", "*.example.com"]);
```

(continues on next page)

(continued from previous page)

```
_paq.push(["trackPageView"]);
```

Tracking multiple domains as one site

To set up tracking between multiple domains, you must use multiple functions: *setDomains* to set a list of domains and *enableCrossDomainLinking* to enable cross domain linking:

```
// specify which domains should be linked
_paq.push(["setDomains", ["*.example.com", "otherdomain.com"]]);

// enable cross domains linking
_paq.push(["enableCrossDomainLinking"]);
```

Note: For cross-domain linking to work, you have to enable link tracking using *enableLinkTracking* function. Remember that links added dynamically to the HTML document won't be tracked unless you call *enableLinkTracking* again. You can learn more about tracking dynamically added links *here*.

Tracking subdirectories of domain as separate websites

To differentiate parts of a website as another site, you must configure JavaScript Tracking Client this way:

```
_paq.push(["setSiteId", "App1"]);
_paq.push(["setTrackerUrl", u+"ppms.php"]);
_paq.push(["trackPageView"]);
```

Afterwards, you can change configuration for selected paths and track them as another site:

```
_paq.push(["setSiteId", "App2"]);

_paq.push(["setCookiePath", "/data/something_useful"]);

_paq.push(["setDomains", "example.com/data/something_useful"]);

_paq.push(["setTrackerUrl", u+"ppms.php"]);
_paq.push(["trackPageView"]);
```

This way, all actions tracked on `/data/something_useful` will be tracked for `App2` instead of `App1`.

If you wish to track a group of pages as separate site, you can use the wildcard in the *setDomains* function.

Collecting page performance metrics

To set up page performance metrics gathering use the *setTimingDataSamplingOnPageLoad* function:

```
// measure performance on 33% of page loads
_paq.push(["setTimingDataSamplingOnPageLoad", 33]);

// track page view and potentially measure page performance
_paq.push(["trackPageView"]);
```

API

The following API allows the user to:

- track page views
- track visits on multiple domains and subdomains
- track e-commerce events (successful orders, cart changes, product and category views)
- track content impressions
- manage custom variables to use them later
- track clicked links to external domains and download files

Table of Contents

- *API*
  - *Command queue*
  - *JavaScript Tracking Client object*
  - *Tracking functions*
    - *Page views*
    - *Custom events*
    - *Goal conversions*
    - *Site search*
    - *E-commerce*
    - *Custom Variables*
    - *Custom Dimensions*
      - *Custom dimensions object*
    - *Content Tracking*
      - *Impressions*
      - *Interactions*
    - *Download and Outlink*
    - *User management*
    - *Cookie management*
    - *Cross domain linking*
    - *JavaScript Tracking Client configuration*
    - *Miscellaneous*

Command queue

Code snippet with tracking code sets up globally accessible command queue `_paq`. Users can issue commands by pushing them onto the command queue with `_paq.push` function. This is the recommended method of calling tracking functions.

_paq.**push**(*command*)
Issues a command, e.g. track page view, custom event, site search etc.

Arguments

- **command** (*Array<string>*) – Array containing a tracking function's *name* followed by its arguments. The number of arguments and their meaning are determined by the tracking function.

Example of usage (tracking a custom event by pushing a command to the command queue):

```
_paq.push(["trackEvent", "video", "video-paused", "intro.mp4", 15.2]);
```

Commands pushed onto the command queue will be executed once the JavaScript Tracking Client loads. After that, _paq.push becomes synchronous, meaning each command is executed at the moment of push.

JavaScript Tracking Client object

JavaScript Tracking Client object offers an alternative method of calling tracking functions. While it's more difficult to access than the *command queue*, it allows to read the return value of a tracking function and makes multi-tracker setups possible.

JavaScript Tracking Client object can be accessed using *Piwik.getTracker* or *Piwik.getAsyncTracker* function.

Piwik.**getTracker**(*trackerUrl*, *siteId*)
Getter for JavaScript Tracking Client object.

Arguments

- **trackerUrl** (*string*) – **Required** URL for JavaScript Tracking Client
- **siteId** (*string*) – **Required** Site ID that will be linked to tracked data.

Returns JavaScript Tracking Client object

Example of usage (accessing JavaScript Tracking Client object and tracking a custom event):

```
var jstc = Piwik.getTracker("https://example.com/", "45e07cbf-c8b3-42f3-a6d6-
↪a5a176f623ef");
jstc.trackEvent("video", "video-paused", "intro.mp4", 15.2);
```

To access internal JavaScript Tracking Client object used for asynchronous tracking you must use the Piwik.getAsyncTracker.

Piwik.**getAsyncTracker**(*trackerUrl*, *siteId*)
Getter for JavaScript Tracking Client instance.

Arguments

- **trackerUrl** (*string*) – **Required** URL for JavaScript Tracking Client
- **siteId** (*string*) – **Required** Site Id that will be linked to tracked data.

Returns JavaScript Tracking Client instance

Example of usage (accessing JavaScript Tracking Client object and tracking a custom event):

```
var jstc = Piwik.getAsyncTracker("https://example.com/", "45e07cbf-c8b3-42f3-a6d6-
↪a5a176f623ef");
jstc.trackEvent("video", "video-paused", "intro.mp4", 15.2);
```

JavaScript Tracking Client object is also accessible through `this` keyword in a special command pushed to command queue, where the first element of the command array is a custom function.

```
_paq.push([function () {
    // *this* is a JavaScript Tracking Client object
    this.addEcommerceItem("01725334", "USB-C chord")
    console.log(this.getEcommerceItems());
}]);
```

Warning: JavaScript Tracking Client object can't be accessed before JavaScript Tracking Client file loads (usually a *ppms.js* file).

Tracking functions

Tracking functions collect and send data to *Collecting & Processing Pipeline*. They can be called on a *JavaScript Tracking Client object* or pushed to the *command queue* as commands.

Page views

trackPageView ([*customPageTitle*])

Tracks page view of the page that the function was run on.

Arguments

- **customPageTitle** (*string*) – **Optional** Custom page title, used only for this event

Example of usage:

Command queue

```
_paq.push(["trackPageView"]);
```

JavaScript Tracking Client object

```
jstc.trackPageView();
```

Note: To overwrite page title for **all events** that will happen on the page (until a reload), use *setDocumentTitle* function.

Note: `trackPageView` is included in the default JavaScript Tracking Client setup snippet. It's likely you're already using it.

Custom events

trackEvent (*category*, *action*[, *name*[, *value*[, *dimensions*]]])

Tracks custom event, e.g. when visitor interacts with the page.

Arguments

- **category** (*string*) – **Required** Event category
- **action** (*string*) – **Required** Event action
- **name** (*string*) – **Optional** Event name
- **value** (*number*) – **Optional** Event value
- **dimensions** (*object*) – **Optional** *Custom dimensions* to pass along with the custom event

Example of usage (tracking when the visitor clicks on the cancel button with exit intent):

Command queue

```
_paq.push(["trackEvent", "Exit intent", "Click on button", "Cancel"]);
```

JavaScript Tracking Client object

```
jstc.trackEvent("Exit intent", "Click on button", "Cancel");
```

Goal conversions

trackGoal (*goalID*[, *conversionValue*[, *dimensions*]])

Tracks manual goal conversion.

Arguments

- **goalID** (*number*|*string*) – **Required** Goal ID (integer or UUID)
- **conversionValue** (*number*) – **Optional** Conversion value (revenue)
- **dimensions** (*object*) – **Optional** *Custom dimensions* to pass along with the conversion

Example of usage (tracking conversion of goal *1* with value *15*):

Command queue

```
_paq.push(["trackGoal", 1, 15]);
```

JavaScript Tracking Client object

```
jstc.trackGoal(1, 15);
```

Site search

trackSiteSearch (*keyword*[, *category*[, *resultCount*[, *dimensions*]]])

Tracks search requests on a website.

Arguments

- **keyword** (*string*) – **Required** What keyword the visitor entered into the search box
- **category** (*string*) – **Optional** Category selected in the search engine, can be set `undefined` if not applicable
- **searchCount** (*number*) – **Optional** The number of search results shown
- **dimensions** (*object*) – **Optional** *Custom dimensions* to pass along with the site search event

Example of usage:

Command queue

```
_paq.push(["trackSiteSearch", "stove", undefined, 20]);
```

JavaScript Tracking Client object

```
jstc.trackSiteSearch("stove", undefined, 20);
```

E-commerce

addEcommerceItem (*productSKU*[, *productName*[, *productCategory*[, *productPrice*[, *productQuantity*]]]])

Adds a product to a virtual shopping cart. If a product with the same SKU is in the cart, it will be removed first. Does not send any data to the *Collecting & Processing Pipeline*.

Arguments

- **productSKU** (*string*) – **Required** Product stock-keeping unit
- **productName** (*string*) – **Optional** Product name
- **productCategory** (*string|Array<string>*) – **Optional** Product category or an array of up to 5 categories
- **productPrice** (*number*) – **Optional** Product price
- **productQuantity** (*number*) – **Optional** The number of units

Example of usage:

Command queue

```
_paq.push(["addEcommerceItem", "craft-311", "Unicorn Iron on Patch", "Crafts &␣
↪Sewing", 499, 3]);
```

JavaScript Tracking Client object

```
jstc.addEcommerceItem("craft-311", "Unicorn Iron on Patch", "Crafts & Sewing",␣
↪499, 3);
```

Note: This function does not send any data to *Collecting & Processing Pipeline*. It only prepares the virtual shopping cart to be sent with *trackEcommerceCartUpdate* or *trackEcommerceOrder*.

Warning: The state of the virtual shopping cart is not persisted in browser storage. You must add all products again after a page reload.

Warning: Adding a product with a SKU that has been previously added will first remove the old product, e.g.:

Command queue

```
_paq.push(["addEcommerceItem", "72625151", "Yellow notebook 150 pages", "School␣
↪supplies", 10.00, 1]); // 1 item with sku 72625151
_paq.push(["addEcommerceItem", "72625151", "Yellow notebook 150 pages", "School␣
↪supplies", 10.00, 2]); // 2 items with sku 72625151, not 3!
```

JavaScript Tracking Client object

```
jstc.addEcommerceItem("72625151", "Yellow notebook 150 pages", "School supplies",␣
↪10.00, 1); // 1 item with sku 72625151
jstc.addEcommerceItem("72625151", "Yellow notebook 150 pages", "School supplies",␣
↪10.00, 2); // 2 items with sku 72625151, not 3!
```

removeEcommerceItem(*productSKU*)

Removes a product with the provided SKU from a virtual shopping cart. If multiple units of that product are in the virtual cart, all of them will be removed. Does not send any data to the *Collecting & Processing Pipeline*.

Arguments

- **productSKU** (*string*) – **Required** stock-keeping unit of a product to remove

Example of usage:

Command queue

```
_paq.push(["removeEcommerceItem", "craft-311"]);
```

JavaScript Tracking Client object

```
jstc.removeEcommerceItem("craft-311");
```

Note: This function does not send any data to *Collecting & Processing Pipeline*. It only prepares the virtual shopping cart to be sent with *trackEcommerceCartUpdate* or *trackEcommerceOrder*.

Warning: The state of the virtual shopping cart is not persisted in browser storage. You must add all products again after a page reload.

clearEcommerceCart()

Removes all items from a virtual shopping cart. Does not send any data to the *Collecting & Processing Pipeline*.

Example of usage:

Command queue

```
_paq.push(["clearEcommerceCart"]);
```

JavaScript Tracking Client object

```
jstc.clearEcommerceCart();
```

Note: This function does not send any data to *Collecting & Processing Pipeline*. It only prepares the virtual shopping cart to be sent with *trackEcommerceCartUpdate* or *trackEcommerceOrder*.

> **Warning:** The state of the virtual shopping cart is not persisted in browser storage. You must add all products again after a page reload.

getEcommerceItems()

Returns a copy of items from a virtual shopping cart. Does not send any data to the *Collecting & Processing Pipeline*.

Returns Object containing all tracked items (format: `Object<productSKU, Array[productSKU, productName, productCategory, price, quantity]>`)

Example of usage:

Command queue

```
_paq.push([function () { console.log(this.getEcommerceItems()); }]);
```

JavaScript Tracking Client object

```
console.log(jstc.getEcommerceItems());
```

Example return value:

```
{
    "52441051": ["52441051", "SUPER Notebook 15\" Ocean Blue", "Laptops", 2200,␣
↪1],
    "19287236": ["19287236", "Earbuds COOL PRO x300 BT", "Accessories", 85, 2],
}
```

> **Warning:** The state of the virtual shopping cart is not persisted in browser storage. You must add all products again after a page reload.

setEcommerceView([*productSKU*[, *productName*[, *productCategory*[, *productPrice*]]]])

Tracks product or category view. Must be followed by a *page view*.

Arguments

- **productSKU** (*string*) – **Optional** Product stock-keeping unit.
- **productName** (*string*) – **Optional** Product name.
- **productCategory** (*string|Array<string>*) – **Optional** Category or an array of up to 5 categories.
- **productPrice** (*number*) – **Optional** Category or an array of up to 5 categories.

When tracking **product views**, provide `productSKU` and optionally other parameters.

When tracking **category views**, provide only `productCategory`. Skip `productSKU`, `productName` and `productPrice` parameters supplying `undefined` where necessary.

Example of usage:

Command queue

```
_paq.push(["setEcommerceView", undefined, undefined, "Crafts & Sewing"]); //␣
↪category view
_paq.push(["trackPageView"]);
```

(continues on next page)

(continued from previous page)

```
_paq.push(["setEcommerceView", "craft-311", "Unicorn Iron on Patch", "Crafts &␣
↪Sewing", 499]); // product view
_paq.push(["trackPageView"]);
```

JavaScript Tracking Client object

```
jstc.setEcommerceView(undefined, undefined, "Crafts & Sewing"); // category view
jstc.trackPageView();

jstc.setEcommerceView("craft-311", "Unicorn Iron on Patch", "Crafts & Sewing",␣
↪499); // product view
jstc.trackPageView();
```

Warning: `setEcommerceView` does not send data itself. It must be followed by a call to *trackPageView*.

trackEcommerceCartUpdate (*cartAmount*)

Tracks items present in a virtual shopping cart (registered with *addEcommerceItem*);

Arguments

- **cartAmount** (*number*) – **Required** The total value of items in the cart

Example of usage:

Command queue

```
_paq.push(["trackEcommerceCartUpdate", 250]);
```

JavaScript Tracking Client object

```
jstc.trackEcommerceCartUpdate(250);
```

Warning: Make sure all products from the cart have been registered using `addEcommerceItem` before tracking a cart update. Remember that when a page is reloaded, the cart resets and all products must be registered again.

trackEcommerceOrder (*orderID*, *orderGrandTotal*[, *orderSubTotal*[, *orderTax*[, *orderShipping*[, *orderDiscount*]]]])

Tracks a successfully placed e-commerce order with items present in a virtual cart (registered using *addEcommerceItem*).

Arguments

- **orderID** (*string*) – **Required** String uniquely identifying an order
- **orderGrandTotal** (*number*) – **Required** Order Revenue grand total - tax, shipping and discount included
- **orderSubTotal** (*number*) – **Optional** Order subtotal - without shipping
- **orderTax** (*number*) – **Optional** Order tax amount
- **orderShipping** (*number*) – **Optional** Order shipping cost
- **orderDiscount** (*number*) – **Optional** Order discount amount

Example of usage:

Command queue

```
_paq.push(["trackEcommerceOrder", "3352", 499, 399, 0, 100]);
```

JavaScript Tracking Client object

```
jstc.trackEcommerceOrder("3352", 499, 399, 0, 100);
```

Warning: `trackEcommerceOrder` function clears the list with registered e-commerce items.

Custom Variables

Deprecated since version 5.5: We strongly advise using custom dimensions instead.

`setCustomVariable` (*index*, *name*[, *value*[, *scope*]])

Sets a custom variable that can be used later.

Arguments

- **`index`** (*`number`*) – **Required** Index from 1 to 5 where the variable is stored
- **`name`** (*`string`*) – **Required** Name of the variable
- **`value`** (*`string`*) – **Optional** Value of the variable, limited to 200 characters
- **`scope`** (*`string`*) – **Optional** Scope of the variable, `"visit"` or `"page"`. The default value is `"visit"`.

Example of usage:

Command queue

```
_paq.push(["setCustomVariable", 1, "AspectRatio", "16:9", "visit"]);
```

JavaScript Tracking Client object

```
jstc.setCustomVariable(1, "AspectRatio", "16:9", "visit");
```

Note: A custom variable with the `"visit"` scope will be saved for an entire session, you don't need to set it on every page.

Warning: Index is separate for each variable scope.

`deleteCustomVariable` (*index*[, *scope*])

Removes a previously set custom variable.

Arguments

- **`index`** (*`number`*) – **Required** Number from 1 to 5 where variable is stored
- **`scope`** (*`string`*) – **Optional** Scope of the variable, `"visit"` or `"page"`. The default value is `"visit"`.

Example of usage:

Command queue

```
_paq.push(["deleteCustomVariable", 1, "visit"]);
```

JavaScript Tracking Client object

```
jstc.deleteCustomVariable(1, "visit");
```

getCustomVariable(*index*[, *scope*])

Returns the value of a previously set custom variable.

Arguments

- **index** (*number*) – **Required** Number from 1 to 5 where variable is stored
- **scope** (*string*) – **Optional** Scope of the variable, `"visit"` or `"page"`. The default value is `"visit"`.

Return type Array[string, string]|boolean

Returns Custom variable value as an array with name and value if the custom variable exists or `false` if it doesn't.

Example of usage:

Command queue

```
_paq.push([function() {
    var customVariable = this.getCustomVariable(1, "visit");
    console.log(customVariable);
}]);
```

JavaScript Tracking Client object

```
var customVariable = jstc.getCustomVariable(1, "visit");
console.log(customVariable);
```

Example return value:

```
["theme", "dark-01"]
```

storeCustomVariablesInCookie()

Enables storing `"visit"` type custom variables in a first party cookie.

Example of usage:

Command queue

```
_paq.push(["storeCustomVariablesInCookie"]);
```

JavaScript Tracking Client object

```
jstc.storeCustomVariablesInCookie();
```

Custom Dimensions

setCustomDimensionValue(*customDimensionID*, *customDimensionValue*)

New in version 15.3.

Sets a custom dimension to be used later.

Arguments

- **customDimensionID** (*number*) – **Required** ID of a custom dimension
- **customDimensionValue** (*string*) – **Required** Value of a custom dimension

Example of usage:

Command queue

```
_paq.push(["setCustomDimensionValue", 3, "loginStatus"]);
```

JavaScript Tracking Client object

```
jstc.setCustomDimensionValue(3, "loginStatus");
```

Warning: When you set a custom dimension, its value will be used in all tracking requests within a page load.

Warning: This function does not send any data to the *Collecting & Processing Pipeline*. It prepares a custom dimension to be sent with following events, e.g. page view, e-commerce events, outlink or download events.

deleteCustomDimension (*customDimensionID*)

Removes a custom dimension with the specified ID.

Arguments

- **customDimensionID** (*number*) – **Required** ID of a custom dimension

Example of usage:

Command queue

```
_paq.push(["deleteCustomDimension", 3]);
```

JavaScript Tracking Client object

```
jstc.deleteCustomDimension(3);
```

getCustomDimensionValue (*customDimensionID*)

New in version 15.3.

Returns the value of a custom dimension with the specified ID.

Arguments

- **customDimensionID** (*number*) – **Required** ID of a custom dimension

Returns Value set with *setCustomDimensionValue*

Return type string

Example of usage:

Command queue

```
_paq.push([function() {
    var customDimension = this.getCustomDimensionValue(3);
    console.log(customDimension);
}]);
```

JavaScript Tracking Client object

```
var customDimension = this.getCustomDimensionValue(3);
```

setCustomDimension (*customDimensionID*, *customDimensionValue*)

Deprecated since version 15.3: Function `setCustomDimension` is deprecated due to the difficulty of use (passed values should be URL encoded). Please use *setCustomDimensionValue* instead.

Sets a custom dimension to be used later.

Arguments

- **customDimensionID** (*number*) – **Required** ID of a custom dimension
- **customDimensionValue** (*string*) – **Required** Value of a custom dimension (should be URL encoded)

Example of usage:

Command queue

```
_paq.push(["setCustomDimension", 3, "loginStatus"]);
```

JavaScript Tracking Client object

```
jstc.setCustomDimension(3, "loginStatus");
```

Warning: When you set a Custom Dimension, that value will be used in all tracking requests within a page load.

Warning: This function does not send any data to the *Collecting & Processing Pipeline*. It sets a Custom Dimension to be sent with following events, e.g. page view, e-commerce events, outlink or download events.

getCustomDimension (*customDimensionID*)

Deprecated since version 15.3: Function `getCustomDimension` is deprecated due to the difficulty of use (returned values are URL-encoded). Please use *getCustomDimensionValue* instead.

Returns the value of a custom dimension.

Arguments

- **customDimensionID** (*number*) – **Required** ID of a custom dimension

Returns Value set with *setCustomDimension*

Return type string

Example of usage:

Command queue

```
_paq.push([ function() {
    var customDimension = this.getCustomDimension(3);
    console.log(customDimension);
}]);
```

JavaScript Tracking Client object

```
var customDimension = jstc.getCustomDimension(3);
console.log(customDimension);
```

Custom dimensions object

Some tracking functions accept an optional `dimensions` parameter. You can use it to pass additional custom dimensions along with the tracked event. Custom dimension object might look like this:

```
{
    "dimension1": "hello",
    "dimension4": "nice%20to%20see%20you",
    "dimension5": "goodbye"
}
```

Warning: Keys in a custom dimension object must be in `"dimensionX"` format, where `X` is the ID of a custom dimension. Keys that don't match this format will be ignored.

Warning: Custom dimension values **must be percent-encoded**. To encode a string, pass it through `encodeURIComponent` function, e.g. `encodeURIComponent("Äpfel?")`.

Content Tracking

Impressions

`trackAllContentImpressions``()`

Scans the entire DOM for content blocks and tracks impressions after all page elements load. It does not send duplicates on repeated calls unless `trackPageView` was called in between `trackAllContentImpressions` invocations.

Example of usage:

Command queue

```
_paq.push(["trackAllContentImpressions"]);
```

JavaScript Tracking Client object

```
jstc.trackAllContentImpressions();
```

`trackVisibleContentImpressions``(`[*checkOnScroll*[, *watchInterval*]]`)`

Scans DOM for all visible content blocks and tracks impressions.

Arguments

- **checkOnScroll** (*boolean*) – **Optional** Whether to scan for visible content on `scroll` event. Default value: `true`.
- **watchInterval** (*number*) – **Optional** Delay, in milliseconds, between scans for new visible content. Periodic checks can be disabled by passing `0`. Default value: `750`.

Example of usage:

Command queue

```
_paq.push(["trackVisibleContentImpressions", true, 2000]);
```

JavaScript Tracking Client object

```
jstc.trackVisibleContentImpressions(true, 2000);
```

Warning: Neither option can be changed after the initial setup.

Warning: `trackVisibleContentImpressions` will not detect content blocks placed in a scrollable element.

trackContentImpressionsWithinNode (*domNode*)

Scans `domNode` (with its children) for all content blocks and tracks impressions.

Arguments

- **domNode** (*Node*) – **Required** DOM node with content blocks (elements with `data-track-content` attribute) inside

Example of usage:

Command queue

```
var element = document.querySelector("#impressionContainer");
_paq.push(["trackContentImpressionsWithinNode", element]);
```

JavaScript Tracking Client object

```
var element = document.querySelector("#impressionContainer");
jstc.trackContentImpressionsWithinNode(element);
```

Note: It can be used with `trackVisibleContentImpressions` to track only visible content impressions.

trackContentImpression (*contentName*, *contentPiece*, *contentTarget*)

Tracks manual content impression event.

Arguments

- **contentName** (*string*) – **Required** Name of a content block
- **contentPiece** (*string*) – **Required** Name of the content that was displayed (e.g. link to an image)
- **contentTarget** (*string*) – **Required** Where the content leads to (e.g. URL of some external website)

Example of usage:

Command queue

```
_paq.push(["trackContentImpression", "promo-video", "https://example.com/public/
↪promo-01.mp4", "https://example.com/more"]);
```

JavaScript Tracking Client object

```
jstc.trackContentImpression("promo-video", "https://example.com/public/promo-01.
↪mp4", "https://example.com/more");
```

logAllContentBlocksOnPage()

Print all content blocks to the console for debugging purposes.

Example output:

```
[
    {
        "name": "promo-video",
        "piece": "https://example.com/public/promo-01.mp4",
        "target": "https://example.com/more"
    }
]
```

Interactions

trackContentInteractionNode(*domNode*[, *contentInteraction*])

Tracks interaction with a block in domNode. Can be called from code placed in `onclick` attribute.

Arguments

- **domNode** (*Node*) – **Required** Node marked as content block or containing content blocks. If content block can't be found, nothing will tracked.
- **contentInteraction** (*string*) – **Optional** Name of interaction (e.g. `"click"`). Default value: `"Unknown"`.

Example of usage:

Command queue

```
var domNode = document.querySelector("#add-image");
_paq.push(["trackContentInteractionNode", domNode, "clicked"]);
```

JavaScript Tracking Client object

```
var domNode = document.querySelector("#add-image");
jstc.trackContentInteractionNode(domNode, "clicked");
```

Example of usage in `onclick` attribute:

```
<button onclick="function(){_paq.push(['trackContentInteractionNode', this,
↪'clicked']);}">Click me!</button>
```

trackContentInteraction(*contentInteraction*, *contentName*, *contentPiece*, *contentTarget*)

Tracks manual content interaction event.

Arguments

- **contentInteraction** (*string*) – **Required** Type of interaction (e.g. `"click"`)
- **contentName** (*string*) – **Required** Name of a content block
- **contentPiece** (*string*) – **Required** Name of the content that was displayed (e.g. link to an image)
- **contentTarget** (*string*) – **Required** Where the content leads to (e.g. URL of some external website)

Example of usage:

Command queue

```
_paq.push(["trackContentInteraction", "clicked", "trackingWhitepaper", "document",
↪ "http://cooltracker.tr/whitepaper"]);
```

JavaScript Tracking Client object

```
jstc.trackContentInteraction("clicked", "trackingWhitepaper", "document", "http://
↪cooltracker.tr/whitepaper");
```

Warning: Use this function in conjunction with `trackContentImpression`, as it can only be mapped with an impression by `contentName`.

Download and Outlink

trackLink (*linkAddress*, *linkType*[, *dimensions*[, *callback*]])

Manually tracks outlink or download event with provided values.

Arguments

- **linkAddress** (*string*) – **Required** URL address of the link
- **linkType** (*string*) – **Required** Type of the link, `"link"` for outlink, `"download"` for download
- **dimensions** (*object*) – **Optional** *Custom dimensions* to pass along with the link event
- **callback** (*function*) – **Optional** Function that should be called after tracking the link

Example of usage:

Command queue

```
_paq.push(["trackLink", "http://www.example.com/example", "link"]);
```

JavaScript Tracking Client object

```
jstc.trackLink("http://www.example.com/example", "link");
```

Example of usage in `onclick` attribute:

```
<button onclick="_paq.push(['trackLink', 'http://www.example.com/example', 'link
↪'])">
    Click me!
</button>
```

enableLinkTracking(*enable*)

Enables or disables automatic link tracking. If enabled, left, right and middle clicks on links will be treated as opening a link. Opening a links to an external site (different domain) creates an outlink event. Opening a link to a downloadable file creates a download event.

Arguments

- **enable** (*boolean*) – **Required** Whether to enable automatic link tracking. The default value is `true`.

Example of usage:

Command queue

```
_paq.push(["trackPageView"]);
_paq.push(["enableLinkTracking"]);
```

JavaScript Tracking Client object

```
jstc.trackPageView();
jstc.enableLinkTracking();
```

Note: `enableLinkTracking` is part of the default Tag Manager's tracking code snippet. It's likely your setup already has it.

Note: Outlinks events are tracked only when a link points to a different (external) domain. If that domain belongs to you and you don't want to track outlinks when visitors open it, use *setDomains* function to define internal domains and subdomains.

Warning: `enableLinkTracking` should be called right after the first `trackPageView` or `trackEvent`.

setIgnoreClasses(*classes*)

Set a list of class names that indicate a link should not be tracked.

Arguments

- **classes** (*string|Array<string>*) – **Required** CSS class name or an array of class names

Example of usage:

Command queue

```
_paq.push(["setIgnoreClasses", ["do-not-track", "ignore-link"]]);
```

JavaScript Tracking Client object

```
jstc.setIgnoreClasses(["do-not-track", "ignore-link"]);
```

Note: Elements with `piwik-ignore` and `piwik_ignore` classes are always ignored.

setLinkClasses (*classes*)
Sets a list of class names that indicate whether a link is an outlink and not download.

Arguments

- **classes** (*string|Array<string>*) – **Required** CSS class name or an array of class names

Example of usage:

Command queue

```
_paq.push(["setLinkClasses", "this-is-an-outlink"]);
```

JavaScript Tracking Client object

```
jstc.setLinkClasses("this-is-an-outlink");
```

Note: Elements with `piwik-link` or `piwik_link` class are always treated as outlinks.

setDownloadClasses (*classes*)
Sets a list of class names that indicate whether a list is a download and not an outlink.

Arguments

- **classes** (*string|Array<string>*) – **Required** CSS class name or an array of class names

Example of usage:

Command queue

```
_paq.push(["setDownloadClasses", "this-is-a-download"]);
```

JavaScript Tracking Client object

```
jstc.setDownloadClasses("this-is-a-download");
```

Note: Elements with `download` attribute, `piwik-download` class or `piwik_download` class are always treated as downloads.

Note: Links containing a *known file extension* will be treated as a downloads as well.

setDownloadExtensions (*extensions*)
Overwrites the list of file extensions indicating that a link is a download.

Arguments

- **extensions** (*string|Array<string>*) – **Required** List of extensions to be set. Can be written as string, e.g. `"zip|rar"`, or an array, e.g. `["zip", "rar"]`.

Links containing a known file extension are treated as downloads and not outlinks. We check for extensions at the end of URL path and in query parameter values. Below are examples of URL with extensions detected.

- http://example.com/path/file**.zip**
- http://example.com/path/file**.zip**#hello

- http://example.com/path/file**.zip**?a=102
- http://example.com/path/?a=file**.zip**
- http://example.com/path/?a=file**.zip**&b=29

The default download extensions list contains the following extensions:

`7z`, `aac`, `apk`, `arc`, `arj`, `asf`, `asx`, `avi`, `azw3`, `bin`, `csv`, `deb`, `dmg`, `doc`, `docx`, `epub`, `exe`, `flv`, `gif`, `gz`, `gzip`, `hqx`, `ibooks`, `jar`, `jpg`, `jpeg`, `js`, `mobi`, `mp2`, `mp3`, `mp4`, `mpg`, `mpeg`, `mov`, `movie`, `msi`, `msp`, `odb`, `odf`, `odg`, `ods`, `odt`, `ogg`, `ogv`, `pdf`, `phps`, `png`, `ppt`, `pptx`, `qt`, `qtm`, `ra`, `ram`, `rar`, `rpm`, `sea`, `sit`, `tar`, `tbz`, `tbz2`, `bz`, `bz2`, `tgz`, `torrent`, `txt`, `wav`, `wma`, `wmv`, `wpd`, `xls`, `xlsx`, `xml`, `z`, `zip`

Example of usage:

Command queue

```
_paq.push(["setDownloadExtensions", "mhj|docx"]);
```

JavaScript Tracking Client object

```
jstc.setDownloadExtensions("mhj|docx");
```

> **Warning:** The list of download extensions is not persisted in the browser. It has to be configured on every page load.

addDownloadExtensions (*extensions*)

Adds new extensions to the download extensions list.

Arguments

- **extensions** (*string|Array<string>*) – **Required** List of extensions to be added. Can be written as string, e.g. `"7z|apk|mp4"`, or an array, e.g. `["7z","apk","mp4"]`.

> **Warning:** The list of download extensions is not persisted in the browser. It has to be configured on every page load.

Example of usage:

Command queue

```
_paq.push(["addDownloadExtensions", "mhj|docx"]);
```

JavaScript Tracking Client object

```
jstc.addDownloadExtensions("mhj|docx");
```

removeDownloadExtensions (*extensions*)

Removes extensions from the download extensions list.

Arguments

- **extensions** (*string|Array<string>*) – **Required** List of extensions to remove. Can be written as string, e.g. `"zip|rar"`, or an array, e.g. `["zip", "rar"]`.

Example of usage:

Command queue

```
_paq.push(["removeDownloadExtensions", "mhj|docx"]);
```

JavaScript Tracking Client object

```
jstc.removeDownloadExtensions("mhj|docx");
```

Warning: The list of download extensions is not persisted in the browser. It has to be configured on every page load.

User management

setUserId(*userID*)

Sets user ID, which will help identify a user of your application across many devices and browsers.

Arguments

- **userID** (*string*) – **Required** Non-empty, unique ID of a user in application

Example of usage:

Command queue

```
_paq.push(["setUserId", "19283"]);
```

JavaScript Tracking Client object

```
jstc.setUserId("19283");
```

resetUserId()

Clears previously set `userID`, e.g. when visitor logs out.

Example of usage:

Command queue

```
_paq.push(["resetUserId"]);
```

JavaScript Tracking Client object

```
jstc.resetUserId();
```

setUserIsAnonymous(*isAnonymous*)

Enables or disables anonymous tracking (anonymous = without consent). Does not send any data to *Collecting & Processing Pipeline*. The next emitted event will have anonymous mode set accordingly.

Arguments

- **isAnonymous** (*boolean*) – **Required** Whether visitor is anonymous

Example of usage:

Command queue

```
_paq.push(["setUserIsAnonymous", true]);
```

JavaScript Tracking Client object

```
jstc.setUserIsAnonymous(true);
```

deanonymizeUser()

Disables anonymous tracking and sends deanonymization event to the *Collecting & Processing Pipeline*. Recommended method for disabling anonymous tracking.

Example of usage:

Command queue

```
_paq.push(["deanonymizeUser"]);
```

JavaScript Tracking Client object

```
jstc.deanonymizeUser();
```

getVisitorId()

Returns 16-character hex ID of the visitor.

Example of usage:

Command queue

```
_paq.push([function () {
    var visitorID = this.getVisitorId();
    console.log(visitorID);
}]);
```

JavaScript Tracking Client object

```
var visitorID = jstc.getVisitorId();
console.log(visitorID);
```

getVisitorInfo()

Returns visitor information.

Return type Array<string>

Returns

String array with the following visitor info:

0. new visitor flag indicating new (`"1"`) or returning (`"0"`) visitor
1. visitor ID (16-character hex number)
2. first visit timestamp (UNIX epoch time)
3. previous visit count (`"0"` for first visit)
4. current visit timestamp (UNIX epoch time)
5. last visit timestamp (UNIX epoch time or `""` if N/A)
6. last e-commerce order timestamp (UNIX epoch time or `""` if N/A)

Example of usage:

Command queue

```
_paq.push([function () {
    var info = this.getVisitorInfo();
    console.log(info);
}]);
```

JavaScript Tracking Client object

```
var info = jstc.getVisitorInfo();
console.log(info);
```

Example output:

```
[
    "0",
    "6d85cb0b727eca52",
    "1624261490",
    "12",
    "1631115486",
    "1631115483",
    "1630590788"
]
```

Cookie management

enableCookies()
Enables all first party cookies. Cookies will be created on the next tracking request.

Note: JavaScript Tracking Client has cookies enabled by default.

---

disableCookies()
Disables all first party cookies. Existing cookies will be deleted in the next page view.

deleteCookies()
Deletes existing tracking cookies on the next page view.

hasCookies()
Returns `true` if cookies are enabled in this browser.

setCookieNamePrefix(*prefix*)
Sets the prefix for analytics tracking cookies. Default is `"_pk_"`.

Arguments

- **prefix** (*string*) – **Required** String that will replace default analytics tracking cookies prefix.

setCookieDomain(*domain*)
Sets the domain for the analytics tracking cookies.

Arguments

- **domain** (*string*) – **Required** Domain that will be set as cookie domain. For enabling subdomain you can use wildcard sign or dot.

setCookiePath(*path*)
Sets the analytics tracking cookies path.

Arguments

- **path** (*string*) – **Required** Path that will be set, default is `"/"`.

setSecureCookie(*secure*)
Toggles the secure cookie flag on all first party cookies (if you are using HTTPS).

Arguments

- **secure** (*boolean*) – **Required** Whether to add secure flag to cookies.

setVisitorCookieTimeout(*seconds*)
Sets the expiration time of visitor cookies.

Arguments

- **seconds** (*number*) – **Required** Number of seconds after which the cookie will expire. Default is 13 months.

setReferralCookieTimeout(*seconds*)
Sets the expiration time of referral cookies.

Arguments

- **seconds** (*number*) – **Required** Number of seconds after which the cookie will expire. Default is 6 months.

setSessionCookieTimeout(*seconds*)
Sets the expiration time of session cookies.

Arguments

- **seconds** (*number*) – **Required** Number of seconds after which the cookie will expire. Default is 30 minutes.

setVisitorIdCookie()
Sets cookie containing *analytics ID*.

Note: It's needed only when JavaScript Tracking Client instance is created without use of *_paq.push()* and script needs to know *analytics ID* before first tracking request is sent. Make sure that it is called after all methods that configure cookie are called (e.g. *setCookieNamePrefix()*, *setCookieDomain()*, *setCookiePath()*, etc.).

Cross domain linking

enableCrossDomainLinking()
Enables cross domain linking. Visitors across domains configured with *setDomains* function will be linked by passing visitor ID parameter in links.

disableCrossDomainLinking()
Disables cross domain linking.

isCrossDomainLinkingEnabled()
Returns boolean telling whether cross domain linking is enabled.

setCrossDomainLinkingTimeout(*seconds*)
Changes the time in which two visits across domains will be linked. The default timeout is 180 seconds (3 minutes).

Arguments

- **seconds** (*number*) – **Required** Number of seconds in which two visits across domains will be linked

getCrossDomainLinkingUrlParameter()

Returns the name of a cross domain URL parameter (query parameter by default) holding visitor ID. This is `"pk_vid"` by default.

Example usage:

Command queue

```
_paq.push([function () {
    var parameter = this.getCrossDomainLinkingUrlParameter();
}]);
```

JavaScript Tracking Client object

```
var parameter = jstc.getCrossDomainLinkingUrlParameter();
```

Note: If your application creates links dynamically, you'll have to add this parameter manually, e.g.

```
var url = "http://myotherdomain.com/path/?" + jstc.
↪getCrossDomainLinkingUrlParameter();
$element.append('<a href="' + url + '">link</a>');
```

---

customCrossDomainLinkDecorator(*urlDecorator*)

Sets custom cross domains URL decorator for injecting visitor ID into URLs. Used when cross domain linking is enabled (see *enableCrossDomainLinking()*).

Arguments

- **urlDecorator** (*function*) – **Required** Function injecting a parameter to a URL address

urlDecorator(*url*, *value*, *name*)

Decorator function accepts link URL, parameter name, parameter value (visitor ID) and returns a URL containing the parameter data.

Arguments

- **url** (*string*) – **Required** Link URL
- **value** (*string*) – **Required** Value of visitor ID that should be passed via URL
- **name** (*string*) – **Required** Name of visitor ID parameter used by JavaScript Tracking Client (can be customized)

Returns Decorated URL or `null` (no change in URL)

Return type string|null

Example of usage (value sent via URL query parameter - equivalent of default implementation):

Command queue

```
_paq.push(["customCrossDomainLinkDecorator", function (url, value, name) {
    var parsedUrl = new URL(url);
    parsedUrl.searchParams.append(name, value);
    return parsedUrl.href;
}]);
```

JavaScript Tracking Client object

```
jstc.customCrossDomainLinkDecorator(function (url, value, name) {
    var parsedUrl = new URL(url);
    parsedUrl.searchParams.append(name, value);
    return parsedUrl.href;
}]);
```

customCrossDomainLinkVisitorIdGetter(*urlParser*)

Sets custom cross domain URL parser for extracting visitor ID from URLs. Should extract data injected by URL decorator (set via *customCrossDomainLinkDecorator()*). The getter should return visitor ID extracted from page URL (used by *enableCrossDomainLinking()*).

Arguments

- **urlParser** (*function*) – **Required** Function extracting a visitor ID from a URL address

urlParser(*url*, *name*)

Parser function accepts page URL, parameter name and returns parameter value (visitor ID).

Arguments

- **url** (*string*) – **Required** Page URL
- **name** (*string*) – **Required** Name of parameter holding visitor ID

Returns Visitor ID value (parsed from URL)

Return type string

Example usage (value sent via URL query parameter - equivalent of default implementation):

Command queue

```
_paq.push(["customCrossDomainLinkVisitorIdGetter", function (url, name) {
    return (new URL(url)).searchParams.get(name) || "";
}]);
```

JavaScript Tracking Client object

```
jstc.customCrossDomainLinkVisitorIdGetter(function (url, name) {
    return (new URL(url)).searchParams.get(name) || "";
});
```

JavaScript Tracking Client configuration

setDomains(*domains*)

Allows to define a list of internal domains. Used in *outlink tracking* for determining whether a link is an outlink and in *cross domain linking* for determining which links should have visitor ID parameter injected.

Arguments

- **domains** (*Array<string>*) – **Required** A list of internal domains. Domains can contain wildcards: `"*"`.

Example of usage:

Command queue

```
_paq.push(["setDomains", ["*.example.com", "*.example.co.uk"]]);
```

JavaScript Tracking Client object

```
jstc.setDomains(["*.example.com", "*.example.co.uk"]);
```

setDocumentTitle(*title*)

Overwrites document title internally. All events sent afterwards will use the provided document title. The title shown in a browser window is not affected.

Arguments

- **title** (*string*) – **Required** Custom title

Example of usage:

Command queue

```
_paq.push(["setDocumentTitle", document.title.toLocaleLowerCase()]);
```

JavaScript Tracking Client object

```
jstc.setDocumentTitle(document.title.toLocaleLowerCase());
```

setTimingDataSamplingOnPageLoad(*sampling*)

Configures page performance data collection. With non-zero sampling (5 by default), some page views will issue a page performance measurement.

Arguments

- **sampling** (*number*) – **Required** Page performance sampling, integer between 0 and 100. 0 disables page performance data collection. 100 measures every page load.

Example of usage:

Command queue

```
_paq.push(["setTimingDataSamplingOnPageLoad", 0]); // disables page performance␣
↪data collection
_paq.push(["setTimingDataSamplingOnPageLoad", 5]); // 5% of page views will by␣
↪followed by a page performance measurement, this is the default behavior
_paq.push(["setTimingDataSamplingOnPageLoad", 30]); // 30% of page views will be␣
↪followed by a page performance measurement
_paq.push(["setTimingDataSamplingOnPageLoad", 100]); // 100% of page views will␣
↪be followed by a page performance measurement
```

JavaScript Tracking Client object

```
jstc.setTimingDataSamplingOnPageLoad(0); // disables page performance data␣
↪collection
jstc.setTimingDataSamplingOnPageLoad(5); // 5% of page views will by followed by␣
↪a page performance measurement, this is the default behavior
jstc.setTimingDataSamplingOnPageLoad(30); // 30% of page views will be followed␣
↪by a page performance measurement
jstc.setTimingDataSamplingOnPageLoad(100); // 100% of page views will be followed␣
↪by a page performance measurement
```

Note: The default sampling value is 5, meaning 5% of page loads will be measured.

> **Warning:** This setting will have an effect only if it's used before the `trackPageView`.

> **Warning:** If a page is closed before it fully loads (e.g. visitor closes the tab immediately after opening the page), page performance data will not be collected.

`getTimingDataSamplingOnPageLoad`()
Returns page performance sampling number.

Example of usage:

Command queue

```
_paq.push([function () {
    console.log(this.getTimingDataSamplingOnPageLoad());
}]);
```

JavaScript Tracking Client object

```
console.log(jstc.getTimingDataSamplingOnPageLoad());
```

Example output:

```
5
```

`enableHeartBeatTimer`()
When a visitor is not producing any events (e.g. because they are reading an article or watching a video), we don't know if they are still on the page. This might skew page statistics, e.g. *time on page* value. *Heartbeat timer* allows us to determine how much time visitors spend on a page by sending heartbeats to the *Collecting & Processing Pipeline* as long as the page is in focus.

Example of usage:

Command queue

```
_paq.push(["enableHeartBeatTimer"]);
```

JavaScript Tracking Client object

```
jstc.enableHeartBeatTimer();
```

Note: The first heartbeat will be sent 15 seconds after the page load. The time between heartbeats increases with the number of heartbeats sent and stops at 5 minutes. When a page looses focus, heartbeats will be paused until the focus is restored. The last heartbeat is sent 30 minutes after the page view.

`setLinkTrackingTimer`(*milliseconds*)
When a visitor produces an events and closes the page immediately afterwards, e.g. when opening a link, the request might get cancelled. To avoid loosing the last event this way, JavaScript Tracking Client will lock the page for a fraction of a second (if wait time hasn't passed), giving the request time to reach the *Collecting & Processing Pipeline*.

`setLinkTrackingTimer` allows to change the default lock/wait time of 500ms.

Arguments

- **milliseconds** (*number*) – **Required** How many milliseconds a request needs to reach the *Collecting & Processing Pipeline*.

Example of usage:

Command queue

```
_paq.push(["setLinkTrackingTimer", 100]);
```

JavaScript Tracking Client object

```
jstc.setLinkTrackingTimer(100);
```

Note: Requests sent using `beacon` method do not lock the page.

Note: Contrary to what the function name suggests, `setLinkTrackingTimer` affects all other types of events. In recent versions of JavaScript Tracking Client, links are sent using `beacon` method if available.

getLinkTrackingTimer()

Returns lock/wait time after a request set by *setLinkTrackingTimer*.

Example of usage:

Command queue

```
_paq.push([function () {
    var time = this.getLinkTrackingTimer();
    console.log(time);
}]);
```

JavaScript Tracking Client object

```
var time = jstc.getLinkTrackingTimer();
console.log(time);
```

setSiteInspectorSetup(*enable*)

Site Inspector is a Chrome browser extension that helps visualize analytics data (e.g. click heat map, scroll map) on tracked pages. Default configuration of JavaScript Tracking Client will add configuration for this extension (in a page HTML), but it is possible to disable this behavior if you don't need it.

Arguments

- **enable** (*boolean*) – **Required** Whether to enable site inspector support.

Example of usage:

Command queue

```
_paq.push(["setSiteInspectorSetup", false]);
```

JavaScript Tracking Client object

```
jstc.setSiteInspectorSetup(false);
```

Miscellaneous

ping()

Ping method sends requests that are not related to any visitor action, but can still update the session. the most common use for this method is updating session custom dimensions or custom variables.

Example of usage:

Command queue

```
_paq.push(["ping"]);
```

JavaScript Tracking Client object

```
jstc.ping();
```

addListener(*domElement*)

Adds automatic link tracking to an HTML element. Can be used to track links added to a document after page load.

Arguments

- **domElement** (*DOMElement*) – **Required** Element that should be tracked like a link.

Example of usage:

Command queue

```
_paq.push(["addListener", document.querySelector("#dynamically-added-link")]);
```

JavaScript Tracking Client object

```
jstc.addListener(document.querySelector("#dynamically-added-link"));
```

setRequestMethod(*method*)

Sets the request method. `GET` and `POST` are valid methods. `GET` is the default.

Arguments

- **method** (*string*) – **Required** Method that will be used in requests. Either `"GET"` or `"POST"`.

Example of usage:

Command queue

```
_paq.push(["setRequestMethod", "POST"]);
```

JavaScript Tracking Client object

```
jstc.setRequestMethod("POST");
```

setRequestContentType(*contentType*)

Sets `Content-Type` header of tracking requests. Used when tracking using `"POST"` method (set by *setRequestMethod*).

Arguments

- **contentType** (*string*) – **Required** Content-Type value to be set.

Example of usage:

Command queue

```
_paq.push(["setRequestContentType", "text/plain"]);
```

JavaScript Tracking Client object

```
jstc.setRequestContentType("text/plain");
```

setCustomRequestProcessing (*function*)
Allows to access and modify query string before sending a page view or ping request.

Arguments

- **function** (*function*) – **Required** Function accepting a query string and returning another query string.

Example of usage:

Command queue

```
_paq.push(["setCustomRequestProcessing", function (query) {
    var modifiedQuery = query.replace("rec=1", "rec=0");
    return modifiedQuery;
}]);
```

JavaScript Tracking Client object

```
jstc.setCustomRequestProcessing(function (query) {
    var modifiedQuery = query.replace("rec=1", "rec=0");
    return modifiedQuery;
});
```

enableJSErrorTracking (*unique*)
Enables tracking of unhandled JavaScript errors.

Arguments

- **unique** (*boolean*) – **Optional** When set to true, tracker will send only unique errors from a page (duplicated errors will be ignored). Default: `true`.

Note: Browsers may limit information about error details if it occurs in script loaded from different origin (see details).

Example of usage:

Command queue

```
_paq.push(["enableJSErrorTracking"]);
```

JavaScript Tracking Client object

```
jstc.enableJSErrorTracking();
```

Reserved names

The following global names are used by JavaScript Tracking Client. Websites that will use this library should avoid using variables with identical names.

- `Piwik`
- `_paq`
- `JSON_PIWIK`
- `piwikPluginAsyncInit`
- `piwikAsyncInit`
- `AnalyticsTracker`
- `piwik_install_tracker`
- `piwik_tracker_pause`
- `piwik_download_extensions`
- `piwik_hosts_alias`
- `piwik_ignore_classes`
- `piwik_log`
- `piwik_track`
- `sevenTag`

2.2 Mobile

2.2.1 Piwik PRO SDK for Android

SDK configuration

Server

- You need a Piwik PRO account on the cloud or an on-premises setup which your mobile app will communicate with. For details, please visit the Piwik PRO website.
- Create a new website (or app) in the Piwik PRO web interface.
- Copy and note the Website ID from "Administration > Websites & apps > Installation" and your server address.

Client

Including the library

Add the JitPack repository to your root `build.gradle` file at the end of repositories:

```
allprojects {
  repositories {
    ...
    maven { url 'https://jitpack.io' }
```

(continues on next page)

(continued from previous page)

```
    }
  }
```

Then add the dependency to the application module `build.gradle` file:

```
  dependencies {
      implementation 'pro.piwik:sdk-framework-android:VERSION'
  }
```

Replace `VERSION` with the latest release name, e.g. `1.0.1`.

Configuration

In order to set up the Piwik PRO tracker, you have two options:

1. Extend `PiwikApplication` class with your Android `Application` class. It forces implementation of one abstract method. That approach is used in the Piwik PRO SDK demo app as below:

```
public class YourApplication extends PiwikApplication{
    @Override
    public TrackerConfig onCreateTrackerConfig() {
        return TrackerConfig.createDefault("https://your.piwik.pro.server.com",
↪"01234567-89ab-cdef-0123-456789abcdef");
    }
}
```

2. Manage the `Tracker` on your own. To configure the `Tracker` you will need a server address and website ID (you can find it in "Administration > Websites & apps > Installation"):

```
public class YourApplication extends Application {
    private Tracker tracker;
    public synchronized Tracker getTracker() {
        if (tracker == null) tracker = Piwik.getInstance(this).newTracker(new␣
↪TrackerConfig("https://your.piwik.pro.server.com", "01234567-89ab-cdef-0123-
↪456789abcdef"));
        return tracker;
    }
}
```

It is not recommended to create multiple `Tracker` instances for the same target as it may lead to over-count of metrics. It is highly recommended to create and manage the tracker in the Application class (to make sure there is only one instance of the tracker). The `Tracker` is thread-safe and can be shared across the application.

```
Tracker tracker = ((PiwikApplication) getApplication()).getTracker();
```

The application is ready to use Piwik PRO SDK.

Using Piwik PRO SDK

It is recommended to use `TrackerHelper` class. It has methods for all common actions, which can be chained in a way that facilitates the correct order and use. Combine it with IDE autocompletion and using the SDK will be more convenient.

For tracking each event with `TrackHelper`, you will need to pass `Tracker` instance. The way of getting the correct `Tracker` instance depends on the configuration option (see section above):

1. Your Android `Application` class extend `PiwikApplication` class

```
Tracker tracker = ((PiwikApplication) getApplication()).getTracker();
```

2. You manage the `Tracker` yourself

```
Tracker tracker = ((YourApplication) getApplication()).getTracker();
```

In further examples we will assume usage of the first option.

Data anonymization

Anonymization is the feature that allows tracking a user's activity for aggregated data analysis even if the user doesn't consent to track the data. If a user does not agree to be tracked, he will not be identified as the same person across multiple sessions.

Personal data will not be tracked during the session (i.e. *user ID*, device ID) If the anonymization is enabled, a new *visitor ID* will be created each time the application starts.

Anonymization is enabled by default.

You can turn the anonymization on and off using the `setAnonymizationState` method:

```
((PiwikApplication) getApplication()).getTracker().setAnonymizationState(false);
```

You can also check the anonymization status using the `isAnonymizationOn` method:

```
((PiwikApplication) getApplication()).getTracker().isAnonymizationOn();
```

Tracking screen views

Requires Analytics

During a valid tracking session, you can track screen views which represent the content the user is viewing in the application. To send a visit on the screen, set the screen path and title on the tracker. This path is internally translated by the SDK to an HTTP URL as the Piwik PRO server uses URLs for tracking views. Additionally, Piwik PRO SDK uses prefixes which are inserted in a generated URL for various types of action(s). For tracking screen views it will use a prefix *screen* by default, however, automatic prefixing can be disabled with the `tracker.setPrefixing(false)` option.

```
public class YourActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Tracker tracker = ((PiwikApplication) getApplication()).getTracker();
        TrackHelper.track().screen("your_activity_path").title("Title").with(tracker);
    }
}
```

- A path (required) – each screen should be mapped to the URL path
- A title (optional) – the title of the action being tracked. It is possible to use slashes (/) to set one or several categories for this action.

To automatically use the activity-stack as a path and activity title as a name, use the overloaded screen method:

```
public class YourActivity extends Activity {
    ...
    TrackHelper.track().screen(YourActivity).with(tracker);
    ...
}
```

- An activity (required) – current instance of android `Activity` class.

In order to bind the tracker to your applications, use the `screens` method. This method will automatically track all open application activities(views) keeping the activity-stack as a path and activity title as the name:

```
TrackHelper.track().screens(getApplication()).with(tracker);
```

Tracking custom events

Requires Analytics

To collect data about the user's interaction with the interactive components of the application, like a button presses or the use of a particular item in the game - use `event` method.

```
TrackHelper.track().event("category", "action").path("/main/actionScreen").name("label
↪").value(1000f).with(tracker);
```

The `track` method allows the specification of the following parameters:

- A category (required) – this String defines the event category. You may define event categories based on the class of user actions (e.g. clicks, gestures, voice commands), or you may define them based on the features available in your application (e.g. play, pause, fast forward, etc.).
- An action (required) – this String defines the specific event action within the category specified. In the example, we are effectively saying that the category of the event is user clicks, and the action is a button click.
- A name (optional) – this String defines a label associated with the event. For example, if you have multiple button controls on a screen, you may use the label to specify the specific view control identifier that was clicked.
- A value (optional) – this Float defines a numerical value associated with the event. For example, if you were tracking "Buy" button clicks, you may log the number of items being purchased or their total cost.
- A path (optional) – the path under which this event occurred.

For more resources, please visit:

- Custom Events Overview
- Ultimate guide to event tracking.

Tracking exceptions

Requires Analytics

Caught exceptions are errors in your app for which you've defined an exception handling code, such as the occasional timeout of a network connection during a request for data. Exceptions are tracked on the server in a similar way as screen views, however, action internally generated for exceptions always use the *fatal* or *caught* prefix, and additionally the *exception* prefix if `tracker.isPrefixing()` this particular option is enabled(true). The URL corresponds to exception stack trace, including the package name, activity path, method name and line number where crash occurred. Bear in mind that Piwik is not a crash tracker therefore use this sparingly.

Measure a caught exception by setting the exception field values on the tracker and sending the hit, as with this example:

```
try {
   // perform action
} catch(Exception ex) {
   TrackHelper.track().exception(ex).description("Content download error").
↪fatal(true).with(tracker);
}
```

- An exception (required) – Caught exception instance.
- A description (optional) – additional information about the issue.
- An isFatal (optional) – true if an exception is fatal.

Tracking social interactions

Requires Analytics

Social interactions such as likes, shares and comments in various social networks can be tracked as below. This, again, is tracked in a similar way as with screen views but the *social* prefix is used when the default `tracker.isPrefixing()` option is enabled.

```
 TrackHelper.track().socialInteraction("Like", "Facebook").target("Game").
↪with(tracker);
```

- An interaction (required) – defines the social interaction, e.g. "Like".
- A network (required) – defines social network associated with interaction, e.g. "Facebook"
- A target (optional) – the target for which this interaction occurred, e.g. "My Piwik PRO app".

The URL corresponds to String, which includes the network, interaction and target parameters separated by slash.

Tracking downloads and app installs

Requires Analytics

You can track the installations and downloads initiated by your application. This only triggers an event once per app version unless you force it. It is recommended to track application install in the Android `Application` class:

```
    TrackHelper.track().download().identifier(new DownloadTracker.Extra.
↪ApkChecksum(this)).with(getTracker());
```

That will use the package name, version and MD5 app checksum as an identifier, e.g. `com.piwikpro.demo:12/7B3DF8ED277BABEA6126C44E9AECEFEA`.

In case you need to specify more parameters, create the instance of the `DownloadTracker` class explicitly:

```
        DownloadTracker downloadTracker = new DownloadTracker(getTracker());
        DownloadTracker.Extra extra = new DownloadTracker.Extra.Custom() {
            @Override
            public boolean isIntensiveWork() {
                return false;
            }
```

(continues on next page)

(continued from previous page)

```
            @Nullable
            @Override
            public String buildExtraIdentifier() {
                return "Demo Android download";
            }
        };

        TrackHelper.track().download(downloadTracker).identifier(extra).force().
↪version("1.0").with(getTracker());
```

- isIntensiveWork() - return true if this should be run async and on a separate thread.
- buildExtraIdentifier() - return a String that will be used as extra identifier or null.

On the analytics panel, all downloads can be viewed in the corresponding section.

Tracking outlinks

Requires Analytics

For tracking outlinks to external websites or other apps opened from your application use the `outlink` method:

```
TrackHelper.track().outlink(new URL("https://www.google.com")).with(getTracker());
```

- A URL (required) – defines the outlink target. HTTPS, HTTP and FTP are are valid.

Tracking search operations

Requires Analytics

Tracking search operations allow the measurement of popular keywords used for various search operations performed inside your application. It can be done via the `search` method:

```
TrackHelper.track().search("Space").category("Movies").count(3).with(getTracker());
```

- A keyword (required) – the searched query that was used in the app.
- A category (optional) – specify a search category.
- A count (optional) – we recommend setting the search count to the number of search results displayed on the results page. When keywords are tracked with a count of 0, they will appear in the "No Result Search Keyword" report.

Tracking content impressions and interactions

Requires Analytics

You can track an impression of an ad in your application as below.

```
TrackHelper.track().impression("Android content impression").piece("banner").target(
↪"https://www.dn.se/").with(getTracker());
```

- A contentName (required) – the name of the content, e.g. "Ad Foo Bar".
- A piece (optional) – the actual content. For instance, the path to an image, video, audio or any text.

- A target (optional) – the target of the content. For instance the URL of a landing page.

Tracking goals

Requires Analytics

By default, goals are defined as "matching" parts of the screen path or screen title. If you want to trigger a conversion manually or track some user interaction, call the method `goal`. Read further about what a goal is in Goal in Piwik PRO.

```
TrackHelper.track().goal(1).revenue(revenue).with(tracker)
```

- A goal (required) – a tracking request will trigger a conversion for the goal of the website being tracked with this ID.
- Revenue (optional) – a monetary value that has been generated as revenue by goal conversion.

Create, view or manage goals is available in the Analytics tab, "Goals" left menu, "Manage goals" section.

Tracking ecommerce transactions

Requires Analytics

If your organization depends on online sales, you need detailed analysis to transform raw e-commerce stats into actionable insights. Revenue, orders, conversion rates, and a host of other product statistics can be analyzed by integrating Piwik with your e-commerce solution.

SDK provides the `order` method that can be used for tracking the orders (including the order items). Sample usage:

```
Tracker tracker = ((YourApplication) getApplication()).getTracker();
EcommerceItems items = new EcommerceItems();
// EcommerceItems.Item("<sku>").name("<product>").category("<category>").price(<cents>
↪).quantity(<number>)
items.addItem(new EcommerceItems.Item("0123456789012").name("Polo T-shirt").category(
↪"Men's T-shirts").price(3000).quantity(2));
items.addItem(new EcommerceItems.Item("0129876543210").name("Leather shoes").category(
↪"Shoes").price(40000).quantity(1));

TrackHelper.track().order("orderId",124144).subTotal(33110).tax(9890).shipping(1000).
↪discount(0).items(items).with(tracker);
```

- `orderId` (required) – a unique String identifying the order
- `grandTotal` (required) – Total amount of the order, in cents
- `subTotal` (optional) – the subTotal (net price) for the order, in cents
- `tax` (optional) – the tax for the order, in cents
- `shipping` (optional) – the shipping for the order, in cents
- `discount` (optional) – the discount for the order, in cents
- `items` (optional) – the items included in the order, use the `EcommerceItems` class to instantiate items

Tracking campaigns

Requires Analytics

Tracking campaigns URLs configured with the online *Campaign URL Builder tool*, allow you to measure how different campaigns (for example with Facebook ads or direct emails) bring traffic to your application. You can track these URLs from the application via the `campaign` method:

```
TrackHelper.track().campaign(new URL("http://example.org/offer.html?pk_campaign=Email-
↪SummerDeals&pk_keyword=LearnMore")).with(getTracker());
```

- A URL (required) – the campaign URL. HTTPS, HTTP and FTP are valid, however, the URL must contain campaign name and keyword parameters.

Tracking custom variables

Requires Analytics

A custom variable is a custom name-value pair that you can assign to your users or screen views, and then visualize the reports of how many visits, conversions, etc. for each custom variable. A custom variable is defined by a name — for example, "User status" — and a value – for example, "LoggedIn" or "Anonymous". It is required for names and values to be encoded in UTF-8.

Each custom variable has a scope. There are two types of custom variables scope - *visit scope* and *screen scope*. The visit scope can be used for any tracking action, and the screen scope can only be applied to tracking screen views.

To set the custom variable of the screen scope, use the `variable` method in the tracking chain:

```
TrackHelper.track()
       .screen("/custom_vars")
       .title("Custom Vars")
       .variable(1, "filter", "price")
       .variable(2, "language", "en")
       .with(getTracker());
```

To use the custom variable of the visit scope, use the `visitVariables` method in the tracking chain:

```
TrackHelper.track()
       .visitVariables(1, "filter", "price")
       .visitVariables(2, "language", "en")
       .event("category", "action")
       .with(tracker);
```

Please note that for the *Default custom variables* option, use the custom variables of the visit scope with indexes 1-3.

Custom variable is defined by three parameters:

- An index (required) – a given custom variable name must always be stored in the same "index" per session. For example, if you choose to store the variable name = "Gender" in index = 1 and you record another custom variable in index = 1, then the "Gender" variable will be deleted and replaced with a new custom variable stored in index 1.
- A name (required) – this String defines the name of a specific Custom Variable such as "User type" (Limited to 200 characters).
- A value (required) – this String defines the value of a specific Custom Variable such as "Customer" (Limited to 200 characters).

Tracking custom dimensions

Requires Analytics

To track a custom name-value pair assigned to your users or screen views, use Custom Dimensions. Note that the custom value data is not sent by itself, but only with other tracking actions such as screen views, events or other tracking action:

```
TrackHelper.track()
       .dimension(1, "visit")
       .dimension(2, "dashboard")
       .screen("Home screen")
       .with(tracker);
```

`1` and `2` are our dimension slots and `visit`, `dashboard` are the dimension values for the tracked screen view.

```
TrackHelper.track()
       .dimension(1, "visit")
       .dimension(2, "billing")
       .event("category", "action")
       .with(tracker);
```

`1` and `2` are our dimension slots and `visit`, `billing` are the dimension values for the tracked event.

Tracking user profile attributes

Requires Audience Manager

The Audience Manager stores visitors' profiles which have data from a variety of sources. One of them can be a mobile application. It is possible to enrich the profiles with more attributes by passing any key-value pair e.g. gender: male, favourite food: Italian, etc. It is recommended to set additional user identifiers such as *email* or *User ID* which will allow the enrichment of existing profiles or merging of profiles rather than creating a new profile. For example, if the user visited the website, performed some actions, filled in a form with his email (his data was tracked and profile created in Audience Manager) and afterwards started using a mobile application, the existing profile will be enriched only if the email was set. Otherwise, a new profile will be created.

For sending profile attributes use `audienceManagerSetProfileAttribute` method:

```
getTracker().setUserMail("john@doe.com");
...
TrackHelper.track().audienceManagerSetProfileAttribute("food", "pizza").add("color",
↪"green").with(getTracker());
```

- A name (required) – defines the profile attribute name (non-null string).
- A value (required) – defines the profile attribute value (non null string).
- An `add` (chain method) – used to specify more attributes to the user within the same event.

Aside from attributes, each event also sends parameters which are retrieved from the tracker instance:

- WEBSITE_ID - always sent,
- USER_ID - if it is set. *Read more* about the User ID,
- EMAIL - if it is set. *Read more* about the email,
- VISITOR_ID - always sent, ID of the mobile application user, generated by SDK

- DEVICE_ID - an Advertising ID that, by default, is fetched automatically when the tracker instance is created. To turn off automatic fetch, use the `setTrackDeviceId(boolean isTracked)` method:

```
getTracker().setTrackDeviceId(false);
```

After calling the `setTrackDeviceId` method, the `DEVICE_ID` variable will not be set if the data anonymization is enabled.

Profile attributes for the user that are tracked will be shown on the Audience Manager - Profile Browser tab.

Audience manager events are dispatched together with analytics events. Therefore, settings set in the tracker for analytics events processing (dispatch interval, cache size and age, etc.) will be same for audience manager events. Once the audience manager event is dispatched, it is no longer stored locally.

Reading user profile attributes

Requires Audience Manager

It is possible to read the attributes of a given profile, however, with some limitations. Due to security reasons (to avoid personal data leakage), it is possible to read only attributes that were enabled for API access (whitelisted) in the Attributes section in Audience Manager. To get user profile attributes use the `audienceManagerGetProfileAttributes` method:

```
        getTracker().audienceManagerGetProfileAttributes(new Tracker.
↪OnGetProfileAttributes() {
            @Override
            public void onAttributesReceived(Map<String, String> attributes) {
                // handle result
            }

            @Override
            public void onError(String errorData) {
                errorData = TextUtils.isEmpty(errorData) ? "Network error": errorData;
                // handle error
            }
        });
```

- An OnGetProfileAttributes (required) – callback to handle request result (call is asynchronous), has two methods `void onAttributesReceived(Map<String, String> attributes)` and `void onError(String errorData)`.
- An attributes (output) – dictionary of key-value pairs, where each pair represents the attribute name (key) and value.
- An errorData (output) – in case of error, only this method will be called. The method passes the error string.

Checking audience membership

Requires Audience Manager

Audiences are allowed to check whether or not the user belongs to a specific group of users defined in the data manger panel based on analytics data and audience manager profile attributes. You can check if the user belongs to a given audience, for example, to show a special offer. To check it, use the `checkAudienceMembership` method:

```
getTracker().checkAudienceMembership(audienceId, new Tracker.
↪OnCheckAudienceMembership() {
            @Override
            public void onChecked(boolean isMember) {
                // handle result
            }

            @Override
            public void onError(String errorData) {
                // handle error
            }
        });
```

- An audienceId (required) – ID of the audience (Audience Manager -> Audiences tab)
- An OnCheckAudienceMembership (required) – callback to handle request result (call is asynchronous), has two methods `void onChecked(boolean isMember)` and `void onError(String errorData)`
- An isMember (output) – a boolean value that indicates if user belongs to audience with given ID
- An errorData (output) – in case of error, only this method will be called. The method passes the error string.

Advanced usage

User ID

UserID will allow the association of events from various sources to the same user. Each time a new visitor enters your page, Piwik PRO assigns a cookie containing a random string of characters. The purpose of this cookie is for Piwik PRO to be able to recognize the same visitor whenever the website is visited again. However, instead of a random string, you can assign your visitors with your own human-friendly name (ex. visitor email). More about UserID. In order to set UserID, use the `setUserId` method:

```
getTracker().setUserId("John Doe");
```

- A UserID (required) – any non-empty unique string identifying the user. Passing null will delete the current UserID

User email address

Used only by Audience Manager

The user email address is an optional parameter for user identification. Similar to UserID, it allows the association of events from various sources to the same user. To set user email use the `setUserMail` method:

```
getTracker().setUserMail("john@doe.com");
```

- A userMail (required) – any non-null string representing email address

Setting up an email helps the Audience Manager to enrich existing profiles or merge profiles which come from other sources (if they also have an email). Check *Tracking user profile attributes* for more information.

Visitor ID

To track user sessions on difference sources, the VisitorID parameter is used. VisitorID is randomly generated when the tracker instance is created, and stored between application launches. It is also possible to reset the VisitorID

manually:

```
tracker.setVisitorId("0123456789abcdef");
```

- A VisitorID (required) – unique visitor ID, must be 16 characters hexadecimal string.

Every unique visitor must be assigned a different ID and this ID must not change after it is assigned. We recommend using UserID instead of VisitorID.

Sessions

A session represents a set of user's interactions with your app. By default, Analytics is closing the session after 30 minutes of inactivity, counting from the last recorded event in session and when the user will open up the app again the new session is started. You can configure the tracker to automatically close the session when users have placed your app in the background for a period of time. That period is defined by the `setSessionTimeout` method.

```
tracker.setSessionTimeout(30 * 60 * 1000);
```

- A timeout (required) – session timeout time in ms.

You can manually start a new session when sending a hit to Piwik by using the `startNewSession` method.

```
tracker.startNewSession();
```

Dispatching

Tracked events are stored temporarily on the queue and dispatched in batches every 30 seconds (default setting). This behavior can be changed with following options:

- `setDispatchInterval(0)` - incoming events will be dispatched immediately
- `setDispatchInterval(-1)` - incoming events will not be dispatched automatically. This lets you gain full control over dispatch process, by using manual dispatch, as in the example below.

```
    Tracker tracker = ((MyApplication) getApplication()).getTracker();
    tracker.setDispatchInterval(-1);
    // Catch and track exception
    try {
        cartItems = getCartItems();
    } catch (Exception e) {
        tracker.trackException(e, e.getMessage(), false);
        tracker.dispatch();
        cartItems = null;
    }
```

In case when more than one event is in the queue, data is sent in bulk (using POST method with JSON payload). It is possible to compress the data before dispatch by using `setDispatchGzipped` method during the app initialization. See the example below for details:

```
    private void initPiwik() {
      ...

        //configure dispatcher to compress JSON with gzip
        getTracker().setDispatchGzipped(true);
```

(continues on next page)

(continued from previous page)

```
    ...
}
```

To take advantage of compressed requests you have to configure HTTP server of the tracker. Use `mod_deflate` (on Apache) or `lua_zlib` (on Nginx). Helpful resources:

- lua_zlib
- lua-nginx-module
- inflate.lua samples

Custom queries

You should be able to use all common actions through the TrackHelper utility, but in some instances, you may want full control over what is sent to the server.

The base method for any event is `track`. You can create your own `TrackMe` objects, set the parameters and then send it:

```
TrackMe trackMe = new TrackMe()
trackMe.set...
/* ... */
Tracker tracker = ((YourApplication) getApplication()).getTracker();
tracker.track(trackMe);
```

Default custom variables

SDK can automatically add information about the platform version, OS version and app version in custom variables with indexes 1-3. By default, this option is turned on. This can be changed via the `setIncludeDefaultCustomVars` method:

```
getTracker().setIncludeDefaultCustomVars(false);
```

In case you need to configure custom variables separately, turn off this option and see the section above regarding tracking custom variables.

Local storage limits

You can set limits for storing events related to maximum size and time for which events are saved in local storage as below. Events older than the set limit will be discarded on the next dispatch attempt. The Piwik backend accepts backdated events for up to 24 hours by default.

To change offline cache age use the `setOfflineCacheAge` method:

```
tracker.setOfflineCacheAge(80085);
```

- A limit (required) – time in ms after which events are deleted, 0 = unlimited, -1 = disabled offline cache. By default, the limit is set to 24 * 60 * 60 * 1000 ms = 24 hours.

You can also specify how large the offline cache may be. If the limit is reached, the oldest files will be deleted first. To change offline cache size use the `setOfflineCacheSize` method:

```
tracker.setOfflineCacheSize(16 * 1000 * 1000);
```

- A limit (required) – size in bytes after which events are deleted, 0 = unlimited. By default, the limit is set to 4 * 1024 * 1024 bytes = 4 Mb.

Opt out

You can enable an app-level opt-out flag that will disable Piwik PRO tracking across the entire app. Note that this flag must be set each time the app starts up and will default to `false`. To set the app-level opt-out, use:

```
getTracker().setOptOut(true);
```

Dry run

The SDK provides a dryRun flag that, when set, prevents any data from being sent to Piwik. The dryRun flag should be set whenever you are testing or debugging an implementation and do not want test data to appear in your Piwik reports. To set the dry run flag, use:

```
getTracker().setDryRunTarget(Collections.synchronizedList(new ArrayList<Packet>()));
```

- A dryRunTarget (required) – a data structure the data should be passed into `List<Packet>` type. Set it to null to disable dry run.

License

Piwik PRO Android SDK is released under the BSD-3 Clause license.

Copyright 2018 Piwik PRO team

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Piwik team nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2.2.2 Piwik PRO SDK for iOS

SDK configuration

Server

- You need a Piwik PRO account on the cloud or an on-premises setup which your mobile app will communicate with. For details, please visit the Piwik PRO website.
- Create a new website (or app) in the Piwik PRO web interface.
- Copy and note the Website ID from "Administration > Websites & apps > Installation" and your server address.

Client

Including the library

Use the following in your Podfile:

```
pod 'PiwikPROSDK', '~> VERSION'
```

Replace `VERSION` with the latest release name, e.g. `'~> 1.0.0'`.

Then run `pod install`. In every file you wish to use the PiwikPROSDK, don't forget to import it.

Configuration

To configure the tracker you will need the URL address of your tracking server and website ID (you can find it in *Administration* > *Websites & apps* > *Installation* on the web interface).

Open the *AppDelegate.m* file and add sdk import:

```
#import <PiwikPROSDK/PiwikPROSDK.h>
```

Configure the tracker with your website ID and URL in the application delegate:

```
- (BOOL)application:(UIApplication *)application␣
↪didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Configure the tracker in your application delegate
    [PiwikTracker sharedInstanceWithSiteID:@"01234567-89ab-cdef-0123-456789abcdef"␣
↪baseURL:[NSURL URLWithString:@"https://your.piwik.pro.server.com"]];
    return YES;
}
```

Using Piwik PRO SDK

SDK supports several different types of actions which can be tracked. If the event dispatch was unsuccessful (network error, server error, etc), the event will be saved in the disk cache and processing will be retried during the next dispatch attempt (in configured dispatch interval). Each event is stored in the disk cache for a specified cache age - the time which defines the maximum time for which event is saved locally.

Data anonymization

Anonymization is the feature that allows tracking a user's activity for aggregated data analysis even if the user doesn't consent to track the data. If a user does not agree to be tracked, he will not be identified as the same person across multiple sessions.

Personal data will not be tracked during the session (i.e. *user ID*, *device ID*) If the anonymization is enabled, a new *visitor ID* will be created each time the application starts.

Anonymization is enabled by default.

You can turn the anonymization on and off by setting the value of the variable `isAnonymizationEnabled`:

```
[PiwikTracker sharedInstance].isAnonymizationEnabled = NO;
```

Tracking screen views

Requires Analytics

The basic functionality of the SDK is the tracking screen views which represent the content the user is viewing in the application. To track a screen you only need to provide the name of the screen. This name is internally translated by the SDK to an HTTP URL as the Piwik PRO server uses URLs for tracking views. Additionally, Piwik PRO SDK uses prefixes which are inserted in generated URLs for various types of action(s). For tracking screen views it will use prefix *screen* by default however automatic prefixing can be disabled with the *isPrefixingEnabled* option. To start tracking screen views, add the following code to your view controllers.

```
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
    [[PiwikTracker sharedInstance] sendView:@"Menu"];
}
```

- A screen name (required) – title of the action being tracked. The appropriate screen path will be generated for this action.

Tracking custom events

Requires Analytics

Custm events can be used to track the user's interaction with various custom components and features of your application, such as playing a song or a video. Category and action parameters are required while the name and value are optional. You can read more about events in the Piwik PRO documentation and ultimate guide to event tracking.

```
[[PiwikTracker sharedInstance] sendEventWithCategory:@"Video" action:@"Play" name:@
↪"Pirates" value:@185];
```

The `sendEventWithCategory` method allows to specify next parameters:

- A category (required) – this String defines the event category. You may define event categories based on the class of user actions (e.g. clicks, gestures, voice commands), or you may define them based upon the features available in your application (e.g. play, pause, fast forward, etc.).
- An action (required) – this String defines the specific event action within the category specified. In the example, we are essentially saying that the category of the event is user clicks, and the action is a button click.

- A name (optional) – this String defines a label associated with the event. For example, if you have multiple button controls on a screen, you might use the label to specify the specific View control identifier that was clicked.
- A value (optional) – this Float defines a numerical value associated with the event. For example, if you were tracking "Buy" button clicks, you might log the number of items being purchased, or their total cost.

Tracking exceptions

Requires Analytics

Tracking exceptions allow the measurement of exceptions and errors in your app. Exceptions are tracked on the server in a similar way as screen views, however, URLs internally generated for exceptions always use the *fatal* or *caught* prefix and, additionally, if the `isPrefixingEnabled` option is enabled, then the additional *exception* prefix is added.

```
[[PiwikTracker sharedInstance] sendExceptionWithDescription:@"Content download error"␣
↪isFatal:YES];
```

- A description (required) – provides the exception message.
- An isFatal (required) – true if an exception is fatal.

Bear in mind that Piwik is not a crash tracker, use this sparingly.

Tracking social interactions

Requires Analytics

Social interactions such as likes, shares and comments in various social networks can be tracked as below. This, again, is tracked in a similar way as screen views but the *social* prefix is used when the default `isPrefixingEnabled` option is enabled.

```
[[PiwikTracker sharedInstance] sendSocialInteractionWithAction:@"like" target:@"Dogs"␣
↪network:@"Facebook"];
```

- An interaction (required) – defines the social interaction, e.g. "Like".
- A network (required) – defines the social network associated with interaction, e.g. "Facebook"
- A target (optional) – the target for which this interaction occurred, e.g. "Dogs".

The URL corresponds to String, which includes network, interaction and target parameters separated by a slash.

Tracking downloads

Requires Analytics

You can track the downloads initiated by your application.

```
[[PiwikTracker sharedInstance] sendDownload:@"http://your.server.com/bonusmap.zip"];
```

- A URL (required) – the URL of the downloaded content.

No prefixes are used for tracking downloads, but each event of this type use an additional parameter `download` whose value equals to specified URL. On the analytics panel all, downloads can be viewed in the corresponding section.

Tracking application installs

Requires Analytics

You can also track installations of your application. This event is sent to the server only once per application version therefore if you wish to track installs, then you can add it in your application delegate immediately after configuring the tracker.

```
- (BOOL)application:(UIApplication *)application␣
↪didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Configure the tracker in your application delegate
    [PiwikTracker sharedInstanceWithSiteID:@"01234567-89ab-cdef-0123-456789abcdef"␣
↪baseURL:[NSURL URLWithString:@"https://your.piwik.pro.server.com"]];
    [[PiwikTracker sharedInstance] sendApplicationDownload];
    return YES;
}
```

Application installation is only tracked during the first launch. In the case of the application being installed but not run, the app installation will not be tracked.

Tracking outlinks

Requires Analytics

For tracking outlinks to external websites or other apps opened from your application use the `sendOutlink` method:

```
[[PiwikTracker sharedInstance] sendOutlink:@"http://great.website.com"];
```

- A URL (required) – defines the outlink target. HTTPS, HTTP and FTP are valid.

Tracking search operations

Requires Analytics

Tracking search operations allow the measurement of popular keywords used for various search operations performed inside your application. It can be done via the `sendSearchWithKeyword` method:

```
[[PiwikTracker sharedInstance] sendSearchWithKeyword:@"Space" category:@"Movies"␣
↪numberOfHits:@42];
```

- `keyword` (required) – the searched query that was used in the app.
- `category` (optional) – specify a search category.
- `numberOfHits` (optional) – we recommend setting the search count to the number of search results displayed on the results page. When keywords are tracked with a count of 0, they will appear in the "No Result Search Keyword" report.

Tracking content impressions and interactions

Requires Analytics

You can track the impression of the ad in your application as below:

```
[[PiwikTracker sharedInstance] sendContentImpressionWithName:@"name" piece:@"piece"␣
↪target:@"target"];
```

When the user interacts with the ad by tapping on it, you can also track it with a similar method:

```
[[PiwikTracker sharedInstance] sendContentInteractionWithName:@"name" piece:@"piece"␣
↪target:@"target"];
```

- A contentName (required) – the name of the content, e.g. "Ad Foo Bar".
- A piece (optional) – the actual content. For instance the path to an image, video, audio, any text.
- A target (optional) – the target of the content e.g. the URL of a landing page.

Tracking goals

Requires Analytics

Goaltracking is used to measure and improve your business objectives. To track goals, you first need to configure them on the server in your web panel. Goals such as, for example, subscribing to a newsletter can be tracked as below with the goal ID that you will see on the server after configuring the goal and optional revenue. The currency for the revenue can be set in the Piwik PRO Analytics settings. You can read more about goals here.

```
[[PiwikTracker sharedInstance] sendGoalWithID:2 revenue:@30];
```

- A goal (required) – tracking request will trigger a conversion for the goal of the website being tracked with this ID.
- `revenue` (optional) – a monetary value that was generated as revenue by this goal conversion.

Tracking ecommerce transactions

Requires Analytics

Ecommerce transactions (in-app purchases) can be tracked to help you improve your business strategy. To track a transaction you must provide two required values - the transaction identifier and `grandTotal`. Optionally, you can also provide values for `subTotal`, `tax`, `shippingCost`, `discount` and list of purchased items as in the example below.

```
[[PiwikTracker sharedInstance] sendTransaction:[PiwikTransaction␣
↪transactionWithBlock:^(PiwikTransactionBuilder *builder) {
    builder.identifier = @"transactionID";
    builder.grandTotal = @5.0;
    builder.subTotal = @4.0;
    builder.tax = @0.5;
    builder.shippingCost = @1.0;
    builder.discount = @0.0;
    [builder addItemWithSku:@"sku1" name:@"bonus" category:@"maps" price:@2.0␣
↪quantity:@1];
    [builder addItemWithSku:@"sku2" name:@"home" category:@"maps" price:@3.0␣
↪quantity:@1];
}]];
```

- An identifier (required) – a unique string identifying the order
- `grandTotal` (required) – The total amount of the order, in cents

- `subTotal` (optional) – the subtotal (net price) for the order, in cents
- `tax` (optional) – the tax for the order, in cents
- `shipping` (optional) – the shipping for the order, in cents
- `discount` (optional) – the discount for the order, in cents
- Items (optional) – the items included in the order, use the `addItemWithSku` method to instantiate items

Tracking campaigns

Requires Analytics

Tracking campaign URLs created with the online Campaign URL Builder tool allow you to measure how different campaigns (for example with Facebook ads or direct emails) bring traffic to your application. You can register a custom URL schema in your project settings to launch your application when users tap on the campaign link. You can track these URLs from the application delegate as below. The campaign information will be sent to the server together with the next analytics event. More details about campaigns can be found in the documentation.

```
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:(NSDictionary␣
↪*)options
{
    return [[PiwikTracker sharedInstance] sendCampaign:url.absoluteString];
}
```

- A URL (required) – the campaign URL. HTTPS, HTTP and FTP are valid - the URL must contain a campaign name and keyword parameters.

Tracking with custom variables

Requires Analytics

To track custom name-value pairs assigned to your users or screen views, you can use custom variables. A custom variable can have a visit scope, which means that they are assigned to the whole visit of the user or action scope meaning that they are assigned only to the next tracked action such as screen view. You can find more information about custom variables here:

It is required for names and values to be encoded in UTF-8.

```
[[PiwikTracker sharedInstance] setCustomVariableForIndex:1 name:@"filter" value:@"lcd
↪" scope:CustomVariableScopeAction];
```

- An index (required) – a given custom variable name must always be stored in the same "index" per session. For example, if you choose to store the variable name = "Gender" in index = 1 and you record another custom variable in index = 1, then the "Gender" variable will be deleted and replaced with new custom variable stored in index 1. Please note that some of the indexes are already reserved. See *Default custom variables* section for details.
- A name (required) – this String defines the name of a specific Custom Variable such as "User type". Limited to 200 characters.
- A value (required) – this String defines the value of a specific Custom Variable such as "Customer". Limited to 200 characters.
- A scope (required) – this String allows the specification of the tracking event type - "visit", "action", etc. The scope is the value from the enum `CustomVariableScope` and could be `CustomVariableScopeVisit` or `CustomVariableScopeAction`.

Tracking with custom dimensions

Requires Analytics

You can also use custom dimensions to track custom values as below. Custom dimensions can also can have a visit or action scope but first have to be defined on the server in your web panel. More details about custom dimensions can be found in the documentation:

```
[[PiwikTracker sharedInstance] setCustomDimensionForIndex:1 value:@"english"␣
↪scope:CustomDimensionScopeVisit];
```

- An index (required) – a given custom dimension must always be stored in the same "index" per session, similar to custom variables. In example `1` is our dimension slot.
- A value (required) – this String defines the value of a specific custom dimension such as "English". Limited to 200 characters.
- A scope (required) – this String allows the specification of the tracking event type - "visit", "action", etc. Scope is the value from enum `CustomDimensionScope` and could be `CustomDimensionScopeVisit` or `CustomDimensionScopeAction`.

Tracking profile attributes

Requires Audience Manager

The Audience Manager stores visitors' profiles, which have data from a variety of sources. One of them can be a mobile application. It is possible to enrich the profiles with more attributes by passing any key-value pair like gender: male, favourite food: Italian, etc. It is recommended to set additional user identifiers such as *email* or *User ID*. This will allow the enrichment of existing profiles or merging profiles rather than creating a new profile. For example, if the user visited the website, browsed or filled in a form with his/her email (his data was tracked and profile created in Audience Manager) and, afterwards started using a mobile application, the existing profile will be enriched only if the email was set. Otherwise, a new profile will be created.

For sending profile attributes use the `sendProfileAttributeWithName` method:

```
[[PiwikTracker sharedInstance] sendProfileAttributeWithName:@"food" value:@"chips"];
```

- A name (required) – defines profile attribute name (non-null string).
- A value (required) – defines profile attribute value (non-null string).

Aside from attributes, each event also sends parameters, that are retrieved from the tracker instance:

- WEBSITE_ID - always sent,
- USER_ID - if It is set. *Read more* about the User ID,
- EMAIL - if It is set. *Read more* about the email,
- VISITOR_ID - always sent, ID of the mobile application user, generated by SDK
- DEVICE_ID - it is a device IDFA, which is not set by default (due to platform limitations). In order to set device ID see *Device ID* section below.

Profile attributes for the user that are tracked will be shown on the Audience Manager - Profile Browser tab.

Reading user profile attributes

Requires Audience Manager

It is possible to read the attributes of a given profile, however, with some limitations. Due to of security reasons to avoid personal data leakage, it is possible to read only attributes that were enabled for API access (whitelisted) in the Attributes section of Audience Manager. To get user profile attributes use the `audienceManagerGetProfileAttributes` method:

```
[[PiwikTracker sharedInstance] audienceManagerGetProfileAttributes:^(NSDictionary␣
↪*profileAttributes, NSError * _Nullable error) {
    // do something with attributes list
}];
```

- `completionBlock` (required) – callback to handle request result (call is asynchronous)
- `profileAttributes` (output) – dictionary of key-value pairs, where each pair represent attribute name (key) and value.
- `errorData` (output) – in case of error only, this method will be called. This method passes the error string.

Checking audience membership

Requires Audience Manager

Audience check allows one to check if the user belongs to a specific group of users defined in the audience manger panel based on analytics data and audience manager profile attributes. You can check if the user belongs to a given audience, for example, to display him/her some type of special offer like in the example below:

```
[[PiwikTracker sharedInstance] checkMembershipWithAudienceID:@"12345678-90ab-cdef-
↪1234-567890abcdef" completionBlock:^(BOOL isMember, NSError * _Nullable error) {
    // do something if is member or handle error
}];
```

- `audienceId` (required) – ID of the audience (Audience Manager -> Audiences tab)
- `completionBlock` (required) – callback to handle request result (call is asynchronous)
- `isMember` (output) – a boolean value that indicates if the user belongs to an audience with a given ID
- `error` (output) – in case of error only, this method will be called. Method pass the error string.

Advanced usage

User ID

The user ID is an additional, optional non-empty unique string identifying the user (not set by default). It can be, for example, a unique username or user's email address. If the provided user ID is sent to the analytics part together with the visitor ID (which is always automatically generated by the SDK), it allows the association of events from various platforms (for example iOS and Android) to the same user provided that the same user ID is used on all platforms. More about UserID. In order to set User ID use `userID` field:

```
[PiwikTracker sharedInstance].userID = @"User Name";
```

- `userID` (required) – any non-empty unique string identifying the user. Passing null will delete the current user ID

User email address

The user email address is another additional, optional string for user identification - if the provided user email is sent to the audience manager part when you send the custom profile attribute configured on the audience manager web panel. Similarly to the user ID, it allows the association of data from various platforms (for example iOS and Android) to the same user as long as the same email is used on all platforms. To set user email use the `userEmail` field:

```
[PiwikTracker sharedInstance].userEmail = @"user@email.com";
```

- A userMail (required) – any non-null string representing email address

It is recommended to set the user email to track audience manager profile attributes as it will create a better user profile.

Visitor ID

SDK uses various IDs for tracking the user. The main one is visitor ID, which is internally randomly generated once by the SDK on the first usage and is then stored locally on the device. The visitor ID will never change unless the user removes the application from the device so that all events sent from his device will always be assigned to the same user in the Piwik PRO web panel. When the anonymization is enabled, a new visitor id is generated each time the application is started. We recommend using userID instead of VisitorID.

Sessions

A session represents a set of user's interactions with your app. By default, Analytics is closing the session after 30 minutes of inactivity, counting from the last recorded event in session and when the user will open up the app again the new session is started. You can configure the tracker to automatically close the session when users have placed your app in the background for a period of time. That period is defined by the `sessionTimeout`:

```
[PiwikTracker sharedInstance].sessionTimeout = 1800
```

- sessionTimeout (required) – session timeout time in seconds. Default: 1800 seconds (30 minutes).

Device ID

The device ID is used to track the IDFA (identifier for advertising). The IDFA is an additional, optional non-empty unique string identifying the device. If you wish to use the IDFA for tracking then you should set the device ID by yourself. Note that if you plan to send your application to the App Store and your application uses IDFA, but does not display ads, then it may be rejected in the App Store review process. You can set the IDFA as in the example below:

```
#import <AdSupport/ASIdentifierManager.h>

NSString *idfa = [[[ASIdentifierManager sharedManager] advertisingIdentifier]␣
↪UUIDString];
[PiwikTracker sharedInstance].deviceID = idfa;
```

Dispatching

All tracking events are saved locally and by default. They are automatically sent to the server every 30 seconds. You can change this interval to any other number as below:

```
[PiwikTracker sharedInstance].dispatchInterval = 60;
```

Gzip compression

You can enable gzip compression for communication with the server as below. By default, requests to the server do not use compression.

```
[PiwikTracker sharedInstance].useGzip = YES;
```

This feature must also be set on server-side using mod_deflate/APACHE or lua_zlib/NGINX (lua_zlib - lua-nginx-module - inflate.lua samples).

Default custom variables

The SDK, by default, automatically adds some information in custom variables about the device (index 1), system version (index 2) and app version (index 3). By default, this option is turned on. This behavior can be disabled with the following setting:

```
[PiwikTracker sharedInstance].includeDefaultCustomVariable = NO;
```

In case you need to configure custom variables separately, turn off this option and see the section above about tracking custom variables.

Local storage limits

You can set limits for storing events related to maximum size and time for which events are saved in local storage. By default, the maximum number of queued events is set to 500 and there is no age limit. It can be changed as below:

```
[PiwikTracker sharedInstance].maxNumberOfQueuedEvents = 100;
[PiwikTracker sharedInstance].maxAgeOfQueuedEvents = 60 * 60 * 24;
```

- `maxNumberOfQueuedEvents` (required) – the maximum number of events after which events in the queue are deleted. By default, the limit is set to 500.
- `maxAgeOfQueuedEvents` (required) – time in ms after which events are deleted. By default, the limit is set to 7 * 24 * 60 * 60 * 1000 ms = 7 days.

Opt-out

You can disable all tracking in the application by using the opt-out feature. No events will be sent to the server if the opt-out is set. By default, opt-out is not set and events are tracked.

```
[PiwikTracker sharedInstance].optOut = YES;
```

License

Piwik PRO iOS SDK is available under the MIT license.

Copyright 2018 Piwik PRO team

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.3 API

2.3.1 Tracking HTTP API

Tracking HTTP API allows sending to analytics information about Visitors page views, events and visits.

2.3.2 Collecting & Processing Pipeline debugger API

Collecting & Processing Pipeline debugger API exposes sessions in live mode. It's a useful tool for verifying JavaScript Tracking Snippet implementation and observing changes done to it.

2.4 Other integrations

2.4.1 Accelerated Mobile Pages integration

Accelerated Mobile Pages (AMP) is an open source framework designed to optimize browsing on mobile devices. This technology can render static content pages much faster than traditional methods. To do that AMP removed the possibility of executing JavaScript on such pages (excluding few approved libraries), so traditional analytic scripts won't work on such pages. You can still measure user engagement using an amp-analytics library.

Basic setup

This setup allows you to track page views. Copy following code to your AMP page while replacing:

- `<INSTANCE_DOMAIN>` - PPAS instance domain (e.g. `analytics.example.com`)
- `<APP_ID>` - PPAS application ID (e.g. `12345678-1234-1234-1234-1234567890ab`)
- `<TRACKER_HASH>` - Cookie hash generated by JavaScript Tracking Client. Check *how to get cookie hash* section for detailed information.

```
<script async custom-element="amp-analytics" src="https://cdn.ampproject.org/v0/amp-
↪analytics-0.1.js"></script>
<amp-analytics type="ppasanalytics">
  <script type="application/json">
  {
```

(continues on next page)

(continued from previous page)

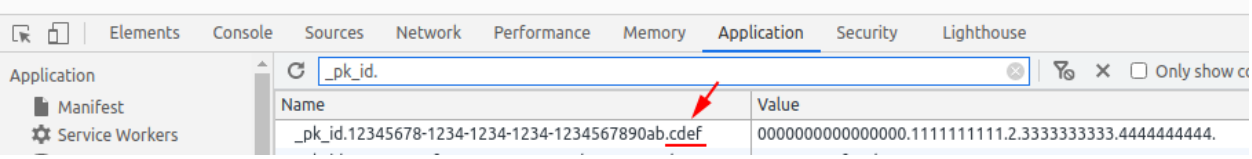
```
    "vars": {
      "host": "<INSTANCE_DOMAIN>",
      "website_id": "<APP_ID>",
      "website_hash": "<TRACKER_HASH>"
    }
  }
  </script>
</amp-analytics>
```

How to get JavaScript Tracking Client cookie hash

If there is no non-AMP page tracked by traditional JavaScript Tracking Client, this value may be removed from configuration or left empty. It's used to guarantee that same cookie will be used by AMP and non-AMP pages on client domain. This value should be taken from the name of the ID cookie generated by JavaScript Tracking Client. Each JavaScript Tracking Client generates unique cookie name based on its configuration. Follow these instructions to get hash from cookie generated by JavaScript Tracking Client:

- Setup JavaScript Tracking Client on non-AMP page (if it was not done already).
- Open tracked page in the browser.
- Open developer tools in the browser and look for cookie starting with `_pk_id.`. Cookie name should look similar to this: `_pk_id.12345678-1234-1234-1234-1234567890ab.cdef`. The part after first dot is the value of App ID of the cookie (if there are multiple cookies starting with `_pk_id.` it may be used to identify correct cookie). After second dot you'll find the cookie hash generated by JavaScript Tracking Client (in the example its value is `cdef`). Copy this part and replace `<TRACKER_HASH>` with it.

Here you can see how to look for JavaScript Tracking Client cookie in Google Chrome developer tools:



Tracking custom events

To track *custom event* you should attach a trigger on the interactive page element and define event values. To do that add to the configuration the triggers section and set up event trigger.

This example will send custom event when page element using "mybutton" ID will be clicked:

```
<amp-analytics type="ppasanalytics">
  <script type="application/json">
  {
    "vars": {
      "host": <instance_domain>,
      "website_id": <app_id>,
      "website_hash": <tracker_hash>
    },
    "triggers": {
      "exampleEvent": {
        "selector": "#mybutton",
        "on": "click",
        "request": "customevent",
```

(continues on next page)

(continued from previous page)

```
        "vars": {
          "event_category": "buttons",
          "event_action": "click",
          "event_name": "testButton"
        }
      }
    }
  }
  </script>
</amp-analytics>
```

These are parameters used by custom event:

- "selector" - CSS selector for element that should be watched
- "on" - HTML event type
- "vars" - Variables that should be used by this event. Custom events expect:
  - "event_category" - required
  - "event_action" - required
  - "event_name" - optional
  - "event_value" - optional

Tracking download events

To track *download event* attach trigger to a link in a similar way to *custom event*.

This example will send download event when page element using "mydownload" ID will be clicked:

```
<amp-analytics type="ppasanalytics">
  <script type="application/json">
  {
    "vars": {
      "host": <instance_domain>,
      "website_id": <app_id>,
      "website_hash": <tracker_hash>
    },
    "triggers": {
      "exampleEvent": {
        "selector": "#mydownload",
        "on": "click",
        "request": "download",
        "vars": {
          "download_url": "https://example.com/whitepaper.pdf"
        }
      }
    }
  }
  </script>
</amp-analytics>
```

These are parameters used by download event:

- "selector" - CSS selector for element that should be watched
- "on" - HTML event type

- "vars" - Variables that should be used by this event. Custom events expect:
  - "download_url" - required

Tracking outlink events

To track *outlink event* attach trigger to a link in a similar way to *custom event*.

This example will send outlink event when page element using "myoutlink" ID will be clicked:

```
<amp-analytics type="ppasanalytics">
  <script type="application/json">
  {
    "vars": {
      "host": <instance_domain>,
      "website_id": <app_id>,
      "website_hash": <tracker_hash>
    },
    "triggers": {
      "exampleEvent": {
        "selector": "#myoutlink",
        "on": "click",
        "request": "outlink",
        "vars": {
          "outlink_url": "https://another-site.com/"
        }
      }
    }
  }
  </script>
</amp-analytics>
```

These are parameters used by outlink event:

- "selector" - CSS selector for element that should be watched
- "on" - HTML event type
- "vars" - Variables that should be used by this event. Custom events expect:
  - "outlink_url" - required

Tracking goal conversions

To track *goal conversion* attach trigger to a link in a similar way to *custom event*.

This example will send goal conversion when page element using "mygoal" ID will be clicked:

```
<amp-analytics type="ppasanalytics">
  <script type="application/json">
  {
    "vars": {
      "host": <instance_domain>,
      "website_id": <app_id>,
      "website_hash": <tracker_hash>
    },
    "triggers": {
      "exampleEvent": {
```

(continues on next page)

(continued from previous page)

```
        "selector": "#mygoal",
        "on": "click",
        "request": "goal",
        "vars": {
          "goal_id": "1",
          "revenue": "59.99"
        }
      }
    }
  }
  </script>
</amp-analytics>
```

These are parameters used by goal event:

- "selector" - CSS selector for element that should be watched
- "on" - HTML event type
- "vars" - Variables that should be used by this event. Custom events expect:
  - "goal_id" - required
  - "revenue" - optional

Track internal search events

To track *internal search event* attach trigger to a link in a similar way to *custom event*.

This example will send internal search event when page element using "mysearch" ID will be clicked:

```
<amp-analytics type="ppasanalytics">
  <script type="application/json">
  {
    "vars": {
      "host": <instance_domain>,
      "website_id": <app_id>,
      "website_hash": <tracker_hash>
    },
    "triggers": {
      "exampleEvent": {
        "selector": "#mysearch",
        "on": "click",
        "request": "search",
        "vars": {
          "search_keyword": "apple",
          "search_category": "fruits",
          "search_result_count": "10",
        }
      }
    }
  }
  </script>
</amp-analytics>
```

These are parameters used by internal search event:

- "selector" - CSS selector for element that should be watched

- "on" - HTML event type
- "vars" - Variables that should be used by this event. Custom events expect:
  - "search_keyword" - required
  - "search_category" - required
  - "search_result_count" - optional

Complete page example

This example shows complete AMP page with 2 buttons. It will send page view, custom event and goal conversion.

```
<!doctype html>
<html amp lang="en">
    <head>
        <meta charset="utf-8">
        <title>AMP example page</title>
        <meta name="viewport" content="width=device-width">
        <link rel="canonical" href="example.html">

        <style amp-boilerplate>body{-webkit-animation:-amp-start 8s steps(1,end) 0s 1␣
↪normal both;-moz-animation:-amp-start 8s steps(1,end) 0s 1 normal both;-ms-
↪animation:-amp-start 8s steps(1,end) 0s 1 normal both;animation:-amp-start 8s␣
↪steps(1,end) 0s 1 normal both}@-webkit-keyframes -amp-start{from{visibility:hidden}
↪to{visibility:visible}}@-moz-keyframes -amp-start{from{visibility:hidden}to
↪{visibility:visible}}@-ms-keyframes -amp-start{from{visibility:hidden}to
↪{visibility:visible}}@-o-keyframes -amp-start{from{visibility:hidden}to
↪{visibility:visible}}@keyframes -amp-start{from{visibility:hidden}to
↪{visibility:visible}}</style><noscript><style amp-boilerplate>body{-webkit-
↪animation:none;-moz-animation:none;-ms-animation:none;animation:none}</style></
↪noscript>

        <script async src="https://cdn.ampproject.org/v0.js"></script>
        <script async custom-element="amp-analytics" src="https://cdn.ampproject.org/
↪v0/amp-analytics-0.1.js"></script>
    </head>
    <body>
        <amp-analytics type="ppasanalytics">
            <script type="application/json">
                {
                    "vars": {
                        "host": "example.piwik.pro",
                        "website_id": "12345678-1234-1234-1234-1234567890ab",
                        "website_hash": "cdef"
                    },
                    "triggers": {
                        "trackRecommendation": {
                            "on": "click",
                            "selector": "#recommend",
                            "request": "customevent",
                            "vars": {
                                "event_category": "social",
                                "event_action": "recommend",
                                "event_name": "News letter"
                            }
                        },
```

(continues on next page)

(continued from previous page)

```
                        "trackSubscription": {
                            "on": "click",
                            "selector": "#subscribe",
                            "request": "goal",
                            "vars": {
                                "goal_id": "1"
                            }
                        }
                    }
                }
            </script>
        </amp-analytics>

        <h1>Welcome</h1>
        <div>
            <button id="recommend">Share this page with friends</button>
        </div>
        <div>
            <button id="subscribe">Subscribe to news letter</button>
        </div>
    </body>
</html>
```

CHAPTER 3

Audience Manager

3.1 Profile data

key	type	description
id	uuid	ID of profile. Example: `"d9a614a1-1234-11ea-a72c-0202c0f2d936"`
website_id	uuid	ID of the website. Example: `"5dff7262-731e-291d-ad23-d1aea83ecd51"`
user_id	string	Value of user id from the Analytics. Example: `"ff1063df11"`
email	string	Email address of the user (detected from submitted form or imported from e.g. CSV). Example: `"test@example.com"`
analytics_visitor_id	string	Analytics ID of the user. Value of cookie analytics_visitor_id. Example: `"b3d31070825871e1"`
analytics_visitor_ids	list	List of analytics_visitor_ids. Example: `["d40bb72cc59e9ef3", ...]`
device_ids	list	List of device IDs. Example:

3.2 JavaScript API

This API provides access to information about users such as ID of *audience* they are part of and their *attributes*. It also allows you to update their *attributes*.

3.2.1 Loading snippet

Add the following snippet on your page to start using this API. It should be added just before the first API usage.

Changed in version 10.0: Loading snippet changed to allow multiple initializations. Now separate scripts can initiate and use this API without interference.

Configuration:

- String `XXX-XXX-XXX-XXX-XXX` should be replaced with *app ID* (e.g. `efcd98a5-335b-48b0-ab17-bf43f1c542be`).
- String `https://your-instance-name.piwik.pro/` should be replaced with your PPAS instance address. (please note that it's used in 2 places in the snippet).

Code:

```
<script>
    (function(a,d,g,h,b,c,e){a[b]=a[b]||{};a[b][c]=a[b][c]||{};if(!a[b][c][e])
↪{a[b][c][e]=function(){(a[b][c][e].q=a[b][c][e].q||[]).push(arguments)};var f=d.
↪createElement(g);d=d.getElementsByTagName(g)[0];f.async=1;f.src=h;d.parentNode.
↪insertBefore(f,d)}})
    (window,document,"script","https://your-instance-name.piwik.pro/audiences/static/
↪widget/audience-manager.api.min.js","ppms","am","api");

    ppms.am.api("create", "XXX-XXX-XXX-XXX-XXX", "your-instance-name.piwik.pro");
</script>
```

This code initializes the API interface in the following ways:

1. Creates a `<script>` tag that asynchronously loads the Audience Manager API library.
2. Initializes the global `ppms.am.api` command queue that schedules commands to be run when the API library is loaded.
3. Schedules `create` command on `ppms.am.api` to initialize the API object with a basic PPAS configuration.

You can use the API command queue (`ppms.am.api`) immediately after step 3.

3.2.2 Command queue

Executing the snippet creates the following global function:

`ppms.am.`**`api`**(*command*, *...args*)

Audience Manager API command queue.

Arguments

- **`command`** (*`string`*) – Command name.
- **`args`** – Command arguments. The number of arguments and their function depend on command.

Returns Commands are expected to be run asynchronously and return no value.

Return type undefined

3.2.3 Commands

All commands work in context of the current *visitor*. Additionally they require communication with a PPAS server and are asynchronous. Callback functions are used to provide response value or information about errors.

Get list of audiences user belongs to

Fetches a list of *audience* IDs the *visitor* belongs to.

Code:

```
ppms.am.api("getAudiences", onFulfilled, onRejected);
```

onFulfilled(*audience_list*)
The fulfilment handler callback (called with result).

Arguments

- **audience_list** (*Array<string>*) – **Required** Array of *audience* IDs the *visitor* belongs to.

  Example:

  ```
  ["e8c6e873-955c-4771-9fd5-92c94577e9d9", "756e5920-422f-4d13-b73a-
  ↪917f696ca288"]
  ```

onRejected(*error_code*)
The rejection handler callback (called with error code).

Arguments

- **error_code** (*string*) – **Required** Error code.

  Example:

  ```
  "server_error"
  ```

Check user membership in the audience

Checks if the *visitor* belongs to the *audience*.

Code:

```
ppms.am.api("checkAudience", audience_id, onFulfilled, onRejected);
```

audience_id
Required `string` ID of the checked *audience*.

Example:

```
"52073260-5861-4a56-be5e-6628794722ee"
```

onFulfilled(*in_audience*)
The fulfilment handler callback (called with result).

Arguments

- **in_audience** (*boolean*) – **Required** *True* when *visitor* is part of the *audience*, *false* otherwise.

  Example:

```
true
```

onRejected(*error_code*)
The rejection handler callback (called with error code).

Arguments

- **error_code** (*string*) – **Required** Error code.

  Example:

```
"server_error"
```

Get user attributes

Fetches the *visitor* profile *attributes*. The *visitor* have to be identified by *analytics ID*.

Note: In order to secure the *PII* data, no *attribute* is returned by default. You need to put each *attribute* you want to access on *attribute whitelist* before it is returned by this command. In order to do that, go to *Audience Manager* > *Attributes* tab and *enable attribute* for the public API access. It is your responsibility to make sure no *visitor PII* data will be available via API.

Code:

```
ppms.am.api("getAttributes", onFulfilled, onRejected);
```

onFulfilled(*attributes*)
The fulfilment handler callback (called with result).

Arguments

- **attributes** (*Object<string,Object<string,(string|number|Array<string>)>>*) – **Required** Object containing *visitor attributes* divided by source.
  - *analytics* - `Object<string,string>` Contains *analytics attributes* about the *visitor* (e.g. browser name, browser version, country).
  - *attributes* - `Object<string,(string|number|Array<string>)>` Contains *custom attributes* about the *visitor* (e.g. first name, last name, email).

  Example:

```
{
    "analytics": {
        "browser_name": "chrome",
        "country": "us"
    },
    "attributes": {
        "favourite_brands": ["Alfa Romeo", "Aston Martin"],
        "age": 32,
        "first_name": "James",
```

(continues on next page)

(continued from previous page)

```
        "last_name": "Bond"
    }
}
```

onRejected(*error_code*)

The rejection handler callback (called with error code).

Arguments

- **error_code** (*string*) – **Required** Error code.

  Example:

```
"server_error"
```

Update user attributes

Creates or updates *visitor custom attributes*.

Note: Any *attribute* can be updated this way whether it is on the *attribute whitelist* or not.

Code:

```
ppms.am.api("updateAttributes", attributes, options);
```

attributes

Required `Object<string,(string|number|Array<string>|object)>` Object containing *attributes* to update:

- key (`string`) - *attribute* name
- value (`string|number|Array<string>|object`) - Value of the *attribute*. System will process it differently depending on its type:
  - `string` - overwrite the *attribute* value with the new value. If the *attribute* was not used before - creates new `text` *attribute*.
  - `number` - overwrite the *attribute* value with the new value. If the *attribute* was not used before - creates new `numeric` *attribute*.
  - `Array<string>` - overwrite the *attribute* value with the new set of values. If the *attribute* was not used before - creates new `text` *attribute* with a list of values.
  - `object` - `ModificationAction` using following format: `{action: string, value: (string|number)}`. It allows to manipulate *attribute* value using one of the following `ModificationAction` `action` values:
    - `"set"` - overwrite *attribute* value using the `ModificationAction` `value`. Works identically to the shorter versions using `string`, `number` or `Array<string>` types.
    - `"add"` - add the `ModificationAction` `value` (or `1`, if not specified) to the *attribute* value.

      Note:

      · Works only on `numeric` *attributes*.

- `ModificationAction value` can be any `number` (including negative and fractional numbers).
- If the *attribute* was not used before - creates new `numeric` *attribute* and sets its value to `0` before performing action.

---

* `"list-add"` - add the `ModificationAction value` to the list of *attribute* values or extend single value *attribute* to a list of values. New value will be a list containing previous value(s) in addition to the added value.

Note:

- Only string values are allowed on the list or can be extended to a list.
- List values are unique. Adding value that already was on the list will not modify the list.

---

* `"list-remove"` - remove the `ModificationAction value` from the list of *attribute* values or delete single value *attribute*. New value will be a list containing previous value(s) without the removed value.

Note:

- Only string values are allowed on the list.

Example:

```
{
    "favourite_color": "black",
    "drink": "Martini",
    "code_number": 7,
    "aliases": ["Peter", "Conrad", "Patrick", "Bill"],
    "kill_count": {
        "action": "add",
        "value": 3,
    },
    "favourite_brands": {
        "action": "list-add",
        "value": "Land Rover",
    },
    "current_missions": {
        "action": "list-remove",
        "value": "Casino Royale",
    },
}
```

`options`

Optional `object` Object that can specify additional *visitor identifiers* and callback functions.

Example:

```
{
    "user_id": user_id,
    "device_id": device_id,
    "email": email,
    "onFulfilled": onFulfilled,
```

(continues on next page)

(continued from previous page)

```
    "onRejected": onRejected
}
```

user_id
Optional `string` If the *application* lets *visitor* sign in - it is possible to pass a unique permanent *user ID* using this parameter. This will let the Audience Manager better identify users across devices (laptop, phone) and sessions.

Example:

```
"jbond"
```

device_id
Optional `string` If the *application* has access to *device ID* - it is possible to pass this value using this parameter. This will let the Audience Manager better identify users across sessions.

Example:

```
"1234567890ABCDEF"
```

email
Optional `string` If the *application* identifies *visitor* via his email - it is possible to pass this value using this parameter. This will let the Audience Manager better identify users across devices (laptop, phone) and sessions.

Example:

```
"j.bond@mi6.gov.uk"
```

onFulfilled()
Optional The fulfilment handler callback (called with result).

onRejected(*error_code*)
Optional The rejection handler callback (called with error code).

Arguments

- **error_code** (*string*) – **Required** Error code.

  Example:

  ```
  "server_error"
  ```

3.3 Form Tracker

Form Tracker gathers data submitted via forms on your page and sends it to the Audience Manager user profile as *attributes*.

Note: Creates or updates user *custom attributes* for each tracker field in the form. The *attribute* name is generated from input tag (HTML tag's `name` attribute or description from its label). Inputs without a name are ignored.

3.3.1 Supported browsers

All modern browsers: Chrome, Firefox, Safari, Edge. Internet Explorer from version 8 and above.

3.3.2 Privacy by design

PPAS follows "Privacy by design" approach to system engineering.

Warning: Form tracker is trying to send its requests using secure **HTTPS** protocol, but **legacy IE browsers** (version 8 and 9) don't have capability to send **CORS** requests using different protocol then the one used by origin page. That means that forms tracked on those browsers will use less secure **HTTP** protocol on pages served via **HTTP** protocol.

Private information

Form Tracker is trying to automatically detect fields containing user's private information and ignores them regardless of the configuration. The following data is never sent to the Audience Manager:

- Value from input with `password` or `hidden` type.
- Credit card number (heuristic detection).
- Credit card validation code (heuristic detection).

Note: Heuristic detection makes best effort to automatically detect and ignore the aforementioned fields, but it does not guarantee success. Additionally, payment forms usually contain more fields with private information (e.g. address) so it is recommended to configure such forms using fields filter.

Configuration

Changed in version 10.0: Loading snippet changed to allow multiple initializations. Tracker will now try to merge configuration of tracked forms as long as `options` will allow it (will be identical).

Changed in version 6.3: Tracked forms are configured using whitelist approach. All forms that should be tracked have to be added to the list, any unrecognized form will be ignored by the tracker. This approach changed from previous blacklist approach where forms had to be included on the list before tracker started ignoring them.

3.3.3 Installation

This section describes how to install the Form Tracker client code on your page.

Using Tag Manager

The Form Tracker tag template is the recommended way to install Form Tracker using PPAS stack.

Manual installation

Add the following snippet on your page to start using Form Tracker.

This code should be added near the top of the `<head>` tag and before any other script or CSS tags. Additionally the snippet has to be configured this way:

- String `XXX-XXX-XXX-XXX-XXX` should be replaced with *app ID* (e.g. `efcd98a5-335b-48b0-ab17-bf43f1c542be`).
- String `https://your-instance-name.piwik.pro//` should be replaced with your PPAS instance address (please note that it's used in 3 places in the snippet).

Changed in version 10.0.

```
<script>
    (function(a,d,g,h,b,c,e){a[b]=a[b]||{};a[b][c]=a[b][c]||{};if(!a[b][c][e])
↪{a[b][c][e]=function(){(a[b][c][e].q=a[b][c][e].q||[]).push(arguments)};var f=d.
↪createElement(g);d=d.getElementsByTagName(g)[0];f.async=1;f.src=h;d.parentNode.
↪insertBefore(f,d)}})
    (window,document,"script","https://your-instance-name.piwik.pro/audiences/static/
↪widget/audience-manager.form.min.js","ppms","am","form");
    ppms.am.form("create", "XXX-XXX-XXX-XXX-XXX", "your-instance-name.piwik.pro",␣
↪forms_config, options);
</script>
```

New in version 6.3.

forms_config
: **Required** `Object<string,(boolean|{type: string, fields: Array<string>})>` Configuration of tracked forms. Default configuration requires that all tracked forms are specified in this object as keys. Each key is another form ID.

  Value of each key can be specified in 2 ways:

  - `true` - All fields in form using this ID will be tracked (this behavior can be changed using *trackingType* option).
  - `Object` - Specifies which fields will be included or excluded from the form.

    type
    : **Required** `"whitelist"|"blacklist"` Defines type of form fields filter.

    fields
    : **Required** `Array<string>` Lists field names used by the filter. Default configuration identifies fields by input `name` attribute, but *useLabels* option can change this behavior.

  Example:

```
{
    "tracked_form": true,
    "form_with_whitelisted_fields": {
        type: "whitelist",
        fields: ["included_field_1", "included_field_2"],
    },
    "form_with_blacklisted_fields": {
        type: "blacklist",
        fields: ["excluded_field_1", "excluded_field_2"],
    },
}
```

New in version 6.3.

options
Optional `object` Options that change behavior of the tracker.

useLabels
Optional `boolean` Defines how tracker identifies form fields. When enabled tracker tries to find label of form field and use its text as identifier. If input doesn't have a label, tracker falls back to default identifier (HTML `name` attribute of the field). Default value: `false`.

Example:

```
false
```

Deprecated since version 6.3.

trackingType
Optional `"whitelist"|"blacklist"` Defines what is default strategy of form configuration. Default value: `"whitelist"`.

- `"whitelist"` - All form IDs that are not set in `forms_config` are ignored by the tracker.
- `"blacklist"` - All form IDs that are set in `forms_config` and use `true` value are ignored by the tracker. Forms defining filtered fields are tracked according to specified fields filter. All other forms are tracked as a whole.

Note: This option is intended for backward compatibility and is planned to be removed in the future.

Example:

```
{
    useLabels: true,
}
```

This code initializes the Form Tracker interface in the following ways:

1. Creates a `<script>` tag that asynchronously loads Audience Manager Form Tracker library.
2. Initializes global `ppms.am.form` command queue that schedules commands to be run when Form Tracker library is loaded.
3. Schedules creation of Form Tracker instance (using `ppms.am.form` function).

3.4 Public HTTP API

3.5 Authorized HTTP API

CHAPTER 4

Consent Manager

4.1 Custom consent form

4.1.1 Enable custom consent form

Consent Manager's JS API enables you to build a custom consent form in place of the default one.

To turn on Custom consent form mode:

1. Go to Administration module
2. Go to "Websites & apps" tab
3. In "Settings" section, find "Privacy" settings
4. Ensure that "Ask visitors for consent" is checked
5. Enable "Use a custom consent form" option

Then you can build a form using the JavaScript API.

4.1.2 Example implementation

Visit Piwik PRO - Custom consent form example page to discover a live demo of Custom consent form implementation (including examples of how to track consent stats using JavaScript API).

4.2 JavaScript API

4.2.1 Introduction

Consent Manager provides a JavaScript API that allows the user to:

- Get compliance types

- Get new compliance types
- Set initial compliance settings
- Set compliance settings
- Get compliance settings
- Send data subject request
- New in version 12.0: Open consent form
- New in version 15.3: Track consent stats

JavaScript API is implemented by providing global JavaScript objects queue responsible for executing command:

`ppms.cm.api`(*command*, *...args*)

Arguments

- **command** (*string*) – Command name
- **args** – Command arguments. The number of arguments and their function depend on command.

Returns Commands are expected to be run asynchronously and return no value

Return type undefined

Consent Manager is fully integrated with Tag Manager. If you already have asynchronous snippet installed, then you are able use Consent Manager's JavaScript API.

4.2.2 Commands

All commands work in the context of the current visitor and website. Additionally, they sometimes require communication with a PPAS server and are asynchronous. Callback functions are used to provide response value or information about errors. `onSuccess(...args)` callback is required, with the exception of `openConsentForm` command where it is optional. `onFailure(exception)` callback is optional and if is specified, any error object occurred will be passed as an argument. If not specified, an error is reported directly on the console output.

Note: For examples of how to use a specific command in your custom consent form implementation (including how to track consent stats), reffer to the Piwik PRO - Custom consent form example

---

Get compliance types

Fetches a list of consent types for the current setup. For the consent type to appear in the output, at least one tag must have it set.

Code:

```
ppms.cm.api('getComplianceTypes', onFulfilled, onRejected);
```

onFulfilled(*types*)

required The fulfillment handler callback (called with result)

Arguments

- **types** (*Array<string>*) – **Required** Array of consent types

  Example:

```
["remarketing", "analytics"]
```

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Get new compliance types

Fetches a list of the consent types which a visitor did not see yet.

Code:

```
ppms.cm.api('getNewComplianceTypes', onFulfilled, onRejected);
```

onFulfilled(*types*)
required The fulfillment handler callback (called with result)

Arguments

- **types** (*Array<string>*) – **Required** Array of consent types

  Example:

  ```
  ["remarketing", "analytics"]
  ```

onRejected(*error*)
The rejection handler callback (called with error code).

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Set initial compliance settings

Sets initial compliance settings (no decision signal for each consent type) in the cookie. Use this command to save "no decision" for the available consent types, to further know that a visitor has seen the form. Result from *getNewComplianceTypes* method can be passed directly.

Code:

```
ppms.cm.api('setInitialComplianceSettings', settings, onFulfilled, onRejected);
```

settings
required The consent settings object

Example:

```
{consents: ['analytics']}
```

or

Example:

```
['analytics']
```

onFulfilled()
required The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Set compliance settings

Set compliance settings based on visitor's decisions. Use this command to save visitor's consent choices from the consent form. Consent Manager forces a page view after the command is invoked, so all tags requiring certain choices will be fired immediately after the consent is given.

Code:

```
ppms.cm.api('setComplianceSettings', settings, onFulfilled, onRejected);
```

settings
required The consent settings object

Example:

```
{consents: {analytics: {status: 1}}}
```

Where `consent.analytics` is consent type and status indicate:

- `0` - user has rejected the consent
- `1` - user has approved the consent

onFulfilled()
required The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Get compliance settings

Returns current privacy settings. Use this command to get visitor's decisions. This command returns an empty object if there were no decisions registered yet.

Code:

```
ppms.cm.api('getComplianceSettings', onFulfilled, onRejected);
```

settings

required The consent settings object

Example:

```
{consents: {analytics: {status: -1, updatedAt: '2018-07-
↪03T12:18:19.957Z'}}}
```

Where `consent.analytics` is consent type and status indicate:

- `-1` - user has not interacted, e.g. has closed a consent popup without any decision
- `0` - user reject consent
- `1` - user approve consent

onFulfilled(*settings*)
required The fulfillment handler callback (called with result)

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Send data subject request

Command that sends a Data subject request to the Consent Manager.

Code:

```
ppms.cm.api('sendDataRequest', request, onFulfilled, onRejected);
```

request
required The subject data request.

Example:

```
{content: 'user input', email: 'example@example.org', type: 'delete_data'}
```

Where `type` is request type, and can be one of:

- `change_data` for data alteration request
- `view_data` for view data request
- `delete_data` for delete data request

onFulfilled()
required The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Open consent form

New in version 12.0.

Command used to open consent form. Works only for built-in consent forms, it will not do anything if Custom consent form mode is enabled.

Code:

```
ppms.cm.api('openConsentForm', onFulfilled, onRejected);
```

onFulfilled(*popupId*, *consentTypes*, *consents*)
The fulfillment handler callback

Arguments

- **popupId** (*string*) – Id of the consent popup

  Example:

  ```
  "ppms_cm_consent_popup_30a851b6-6bf4-45f9-9a53-583401bb5d60"
  ```

- **consentTypes** (*array<string>*) – Array of consent types

  Example:

  ```
  ["analytics", "conversion_tracking", "remarketing"]
  ```

- **consents** (*array<string>*) – Array list of all given consents

  Example:

  ```
  ["analytics", "remarketing"]
  ```

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Track Main Form view

New in version 15.3.

Command used to track Consent Form main view (automatic view, when user enters the website for the first time).

Code:

```
ppms.cm.api('trackMainFormView', onFulfilled, onRejected);
```

onFulfilled()
The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Track Reminder Widget view

New in version 15.3.

Command used to track Consent Form view caused by clicking on Reminder Widget.

Code:

```
ppms.cm.api('trackReminderWidgetView', onFulfilled, onRejected);
```

onFulfilled()
The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Track Privacy Policy Link view

New in version 15.3.

Command used to track Consent Form view caused by clicking on Privacy Policy Link.

Code:

```
ppms.cm.api('trackPrivacyPolicyLinkView', onFulfilled, onRejected);
```

onFulfilled()
The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Track *Agree to all* click

New in version 15.3.

Command used to track clicks on the *Agree to all* button.

Code:

```
ppms.cm.api('trackAgreeToAllClick', onFulfilled, onRejected);
```

onFulfilled()
The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Track *Reject all* click

New in version 15.3.

Command used to track clicks on the *Reject all* button.

Code:

```
ppms.cm.api('trackRejectAllClick', onFulfilled, onRejected);
```

onFulfilled()
The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Track *Save choices* click

New in version 15.3.

Command used to track clicks on the *Save choices* button.

Code:

```
ppms.cm.api('trackSaveChoicesClick', onFulfilled, onRejected);
```

onFulfilled()
The fulfillment handler callback

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

Track close button click

New in version 15.3.

Command used to track clicks on the close button (*X*).

Code:

```
ppms.cm.api('trackCloseButtonClick', onFulfilled, onRejected);
```

onFulfilled()
The fulfillment handler callback

onRejected(*error*)

The rejection handler callback (called with error code). If not specified, the exception will be thrown in the main stack trace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception

CHAPTER 5

Tag Manager

5.1 Authorized HTTP API

5.1.1 Tags

5.1.2 Triggers

5.1.3 Variables

5.1.4 Versions

5.1.5 Changelog

5.1.6 Operations

5.2 Custom data layer name

A data layer is a data structure on your site or app where you can store data and access it with Tag Manager. The default data layer name is `dataLayer`, but you can change it to a custom one.

5.2.1 Choose a unique data layer name

Use a unique data layer name. Make sure that it's not used by other tools installed on your site or app. If the names are the same, the tools can interfere with each other.

To check if the data layer name is used on your site or app, follow these steps:

1. Pick your new data layer name. Example: `customDataLayer`.
2. In a browser's console, run the following script with the picked name:

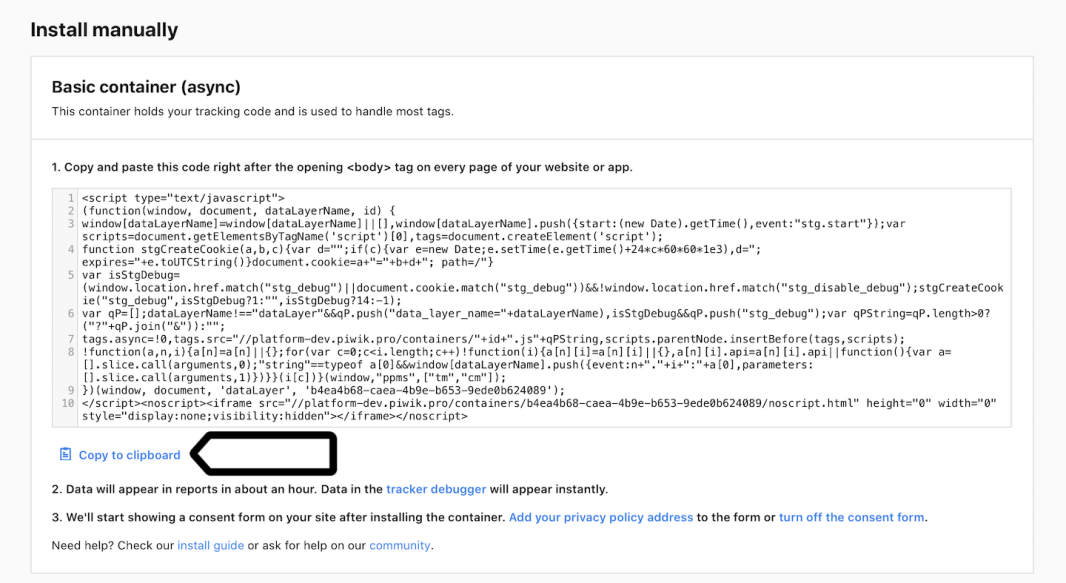
```
var dataLayerName = "customDataLayer";
!window.hasOwnProperty(dataLayerName);
```

3. If the return statement is `true`, you can use this name safely. It means that no other tool is using this name.

5.2.2 Rename your data layer

To rename the data layer, follow these steps:

1. Log in to Piwik PRO.
2. Go to **Menu > Administration**.
3. Navigate to **Sites & apps**.
4. Select the site or app for which you want to rename the data layer.
5. Navigate to **Installation**.
6. Copy the basic container's code. You'll modify this code in the next steps.

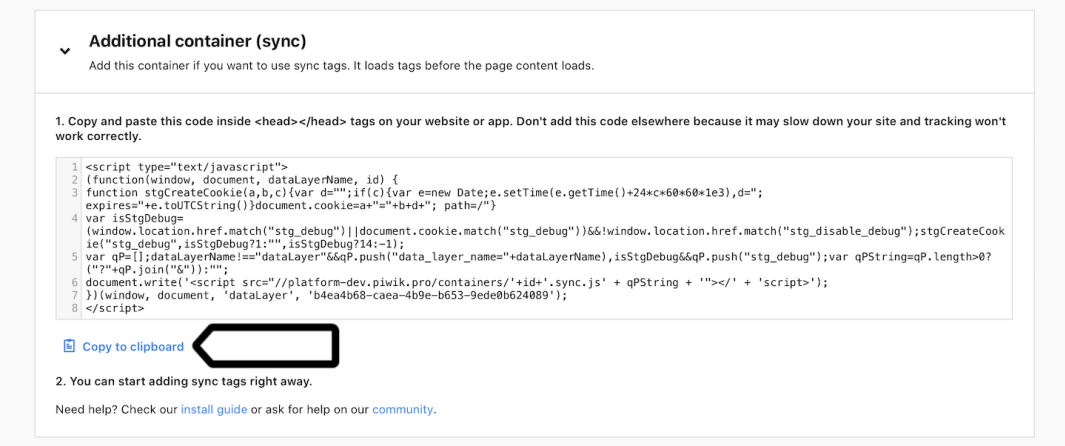


7. In the copied code, change `dataLayer` to a custom name.

```
(window, document, 'dataLayer', '69bc995f-c40a-42ae-b756-b8b9fbc16508');
```

```
<script type="text/javascript">
(function(window, document, dataLayerName, id) {
window[dataLayerName]=window[dataLayerName]||[],window[dataLayerName].push({start:(new Date).getTime(),event:"stg.start"});var
scripts=document.getElementsByTagName('script')[0],tags=document.createElement('script');
function stgCreateCookie(a,b,c){var d="";if(c){var e=new Date;e.setTime(e.getTime()+24*c*60*60*1e3),d=";
expires="+e.toUTCString()}document.cookie=a+"="+b+d+"; path=/"}
var isStgDebug=
(window.location.href.match("stg_debug")||document.cookie.match("stg_debug"))&&!window.location.href.match("stg_disable_debug");stgCreateCooki
e("stg_debug",isStgDebug?1:"",isStgDebug?14:-1);
var qP=[];dataLayerName!=="dataLayer"&&qP.push("data_layer_name="+dataLayerName),isStgDebug&&qP.push("stg_debug");var qPString=qP.length>0?
("?"+qP.join("&")):"";
tags.async=!0,tags.src="https://clearbank.containers.piwik.pro/"+id+".js"+qPString,scripts.parentNode.insertBefore(tags,scripts);
!function(a,n,i){a[n]=a[n]||{};for(var c=0;c<i.length;c++)!function(i){a[n][i]=a[n][i]||{},a[n][i].api=a[n][i].api||function(){var a=
[].slice.call(arguments,0);"string"==typeof a[0]&&window[dataLayerName].push({event:n+"."+i+":"+a[0],parameters:[].slice.call(arguments,1)})}}
(i[c])}(window,"ppms",["tm","cm"]);
})(window, document, 'dataLayer', '69bc995f-c40a-42ae-b756-b8b9fbc16508');
</script><noscript><iframe src="https://clearbank.containers.piwik.pro/69bc995f-c40a-42ae-b756-b8b9fbc16508/noscript.html" height="0"
width="0" style="display:none;visibility:hidden"></iframe></noscript>
```

8. Paste the code right after the opening **<body>** tag on every page of your website or app.
9. Optionally, copy the additional container's code. You'll modify this code in the next steps.



10. In the copied code, change `dataLayer` to a custom name.

```
(window, document, 'dataLayer', '69bc995f-c40a-42ae-b756-b8b9fbc16508');
```

```
<script type="text/javascript">
(function(window, document, dataLayerName, id) {
function stgCreateCookie(a,b,c){var d="";if(c){var e=new Date;e.setTime(e.getTime()+24*c*60*60*1e3),d=";
expires="+e.toUTCString()}document.cookie=a+"="+b+d+"; path=/"}
var isStgDebug=
(window.location.href.match("stg_debug")||document.cookie.match("stg_debug"))&&!window.location.href.match("stg_disable_debug");stgCreateCooki
e("stg_debug",isStgDebug?1:"",isStgDebug?14:-1);
var qP=[];dataLayerName!=="dataLayer"&&qP.push("data_layer_name="+dataLayerName),isStgDebug&&qP.push("stg_debug");var qPString=qP.length>0?
("?"+qP.join("&")):"";
document.write('<script src="http                    ers.piwik.pro/'+id+'.sync.js' + qPString + '"><\/' + 'script>');
})(window, document, 'dataLayer'                    756-b8b9fbc16508');
</script>
```

Note: If you're using both containers, use the same data layer name in each container. Otherwise, things can break.

11. Paste the code inside **<head></head>** tags on your website or app. Don't add this code elsewhere because it may slow down your site and tracking won't work correctly.
12. Replace all existing references to the old data layer name. For example, if you use

```
dataLayer.push({event: "test-event"});
```

replace it with

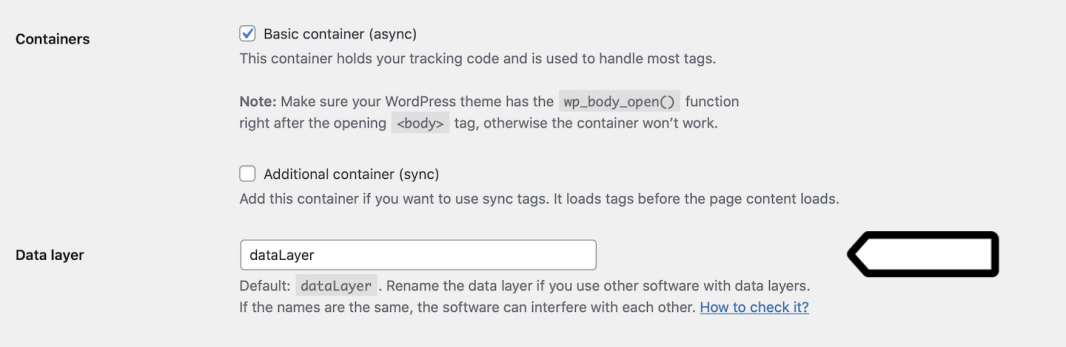
```
customDataLayer.push({event: "test-event"});
```

5.2.3 WordPress plugin: rename your data layer

If you installed our containers with the WordPress plugin, you can quickly rename the data layer in the plugin settings.

To rename the data layer in our WordPress plugin, follow these steps:

1. In your WordPress admin panel, go to **Settings > Piwik PRO**.
2. In **Data layer**, change the name to a custom one.



3. Click **Save changes**.
4. Replace all existing references to the old data layer name. For example, if you use

```
dataLayer.push({event: "test-event"});
```

   replace it with

```
customDataLayer.push({event: "test-event"});
```

5.3 Content Security Policy (CSP)

5.3.1 Introduction

Specifying Content Security Policy is a common way to secure web applications. This mechanism allows specifying which scripts and styles can execute on page. It can be done either by adding a `Content-Security-Policy` header or an appropriate meta tag.

You can read about Consent Security Policy here: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

5.3.2 Content Security Policy nonce configuration

It is common to allow only scripts and styles that were received from known domains or ones that have special nonce attribute. Nonce mechanism relies on two steps, defining nonce value in Content Security Policy and placing nonce value as an attribute in styles and scripts.

Defining nonce in Content Security Policy settings

Nonce mechanism requires additional definition in `script-src` directive of Content Security Policy:

```
script-src <your-sources> 'nonce-INSERT_VALID_NONCE_VALUE';
```

Note: Nonce value should be generated on the server-side. Its value should be different for each request. Please note that we leave here space for your permitted sources **<your-sources>**.

Adding nonce to container code

Consequently, default container code requires following modifications to work:

- **asynchronous snippet** - given container code following changes (highlighted) are required:

```
<script type="text/javascript" nonce="INSERT_VALID_NONCE_VALUE">
    (function(window, document, dataLayerName, id) {
    window[dataLayerName]=window[dataLayerName]||[],window[dataLayerName].push(
↪{start:(new Date).getTime(),event:"stg.start"});
    var scripts=document.getElementsByTagName('script')[0],tags=document.
↪createElement('script');
    function stgCreateCookie(a,b,c){var d="";if(c){var e=new Date;e.setTime(e.
↪getTime()+24*c*60*60*1e3),d=";expires="+e.toUTCString()}document.cookie=a+"=
↪"+b+d+"; path=/"}
    var isStgDebug=(window.location.href.match("stg_debug")||document.cookie.
↪match("stg_debug"))&&!window.location.href.match("stg_disable_debug");
    stgCreateCookie("stg_debug",isStgDebug?1:"",isStgDebug?14:-1);
    var qP=[];dataLayerName!=="dataLayer"&&qP.push("data_layer_name=
↪"+dataLayerName),isStgDebug&&qP.push("stg_debug");
    var qPString=qP.length>0?("?"+qP.join("&")):"";
    tags.async=!0,tags.src="//client.containers.piwik.pro/"+id+".js"+qPString,
    scripts.parentNode.insertBefore(tags,scripts);
    !function(a,n,i){a[n]=a[n]||{};for(var c=0;c<i.length;c++)!function(i)
↪{a[n][i]=a[n][i]||{},a[n][i].api=a[n][i].api||function(){
    var a=[].slice.call(arguments,0);"string"==typeof a[0]&&window[dataLayerName].
↪push({event:n+"."+i+":"+a[0],parameters:[].slice.call(arguments,1)})}}(i[c])}
↪(window,"ppms",["tm","cm"]);
    })(window, document, 'dataLayer', 'feacd61d-0232-40a1-96c3-7e469f7bfa7f');
</script>
<noscript>
    <iframe src="//client.containers.piwik.pro/feacd61d-0232-40a1-96c3-
↪7e469f7bfa7f/noscript.html" height="0" width="0" style="display:none;
↪visibility:hidden"></iframe>
</noscript>
```

- **synchronous snippet** - following changes (highlighted) are required:

```
<script type="text/javascript" nonce="INSERT_VALID_NONCE_VALUE">
    (function(window, document, dataLayerName, id) {
    function stgCreateCookie(a,b,c){var d="";if(c){var e=new Date;e.setTime(e.
↪getTime()+24*c*60*60*1e3),d=";expires="+e.toUTCString()}document.cookie=a+"=
↪"+b+d+"; path=/"}
    var isStgDebug=(window.location.href.match("stg_debug")||document.cookie.
↪match("stg_debug"))&&!window.location.href.match("stg_disable_debug");
    stgCreateCookie("stg_debug",isStgDebug?1:"",isStgDebug?14:-1);
    var qP=[];dataLayerName!=="dataLayer"&&qP.push("data_layer_name=
↪"+dataLayerName),isStgDebug&&qP.push("stg_debug");
    var qPString=qP.length>0?("?"+qP.join("&")):"";
    document.write('<script src="//client.containers.piwik.pro/'+id+'.sync.js' +␣
↪qPString + '" nonce="INSERT_VALID_NONCE_VALUE"></' + 'script>');
    })(window, document, 'dataLayer', 'feacd61d-0232-40a1-96c3-7e469f7bfa7f');
</script>
```

Note: All that is needed for Tag Manager to work is to replace **INSERT_VALID_NONCE_VALUE** with generated nonce value. It should be done twice for both asynchronous and synchronous snippet.

5.3.3 Adjusting tags to work with Content Security Policy

- **asynchronous tags** - in most cases there should not be any change required to make asynchronous tags work. Tag Manager will automatically insert nonce attribute to all fired tags. Only exceptions is when Your tag adds other scripts/styles on page by itself - in such case, You should add nonce attribute manually.
- **synchronous tags** - since synchronous tags have to fire before whole page is loaded, following procedure is recommended:

  1. Create new variable with value of nonce parameter. It is not required to create nonce variable in admin panel. Just pushing it on dataLayer before script is executed is enough.

```
window.dataLayer.push({
    nonce: INSERT_VALID_NONCE_VALUE
});
```

  2. Use created variable as value for nonce attribute like follows:

```
<script nonce="{{ nonce }}">
    console.log("I'm synchronous tag!");
    document.write('<p id="synchronous-tag">I was inserted by synchronous tag
↪</p>');
</script>
```

Note: Finally, not all 3rd party tools that are available as built-in templates are adjusted to work with Content Security Policy. This includes e.g. Google Analytics. In such cases, please refer to documentation of each respective tool (e.g. https://developers.google.com/web/fundamentals/security/csp).

5.3.4 Tag Manager debugger

To load all necessary assets from Tag Manager debugger you need to define source with `img-src`, `font-src` and `style-src` directives:

```
img-src <your-sources> client.containers.piwik.pro;
font-src <your-sources> client.containers.piwik.pro;
style-src <your-sources> client.containers.piwik.pro;
```

5.3.5 Consent Manager form assets

If your website is GDPR compliant then you need to describe `connect-src`, `style-src` and `img-src` directives:

```
connect-src <your-sources> client.piwik.pro client.containers.piwik.pro;
style-src <your-sources> 'nonce-INSERT_VALID_NONCE_VALUE';
```

Note: Please note that we define here tracking domain **client.piwik.pro** for collecting visitor consents and container domain **client.containers.piwik.pro** for fetching consent form assets.

5.3.6 Consent Manager's data subject request widget

When using a data subject request widget, you need to add a nonce attribute to its `<script>` tag.

```
<div id="ppms_cm_data_subject" class="ppms_cm_data_subject_widget__wrapper" data-
↪editor-centralize="true" data-main-container="true" data-root="true">
    <h3 id="ppms_cm_data_subject_header" class="header3">Data requests</h3>
    <p id="ppms_cm_data_subject_paragraph" class="paragraph">
        Please select below the type of data request along with any special requests␣
↪in the body of the message. (...)
    </p>
    <form id="ppms_cm_data_subject_form" class="ppms_cm_data_subject_form" data-
↪disable-select="true">
        ...
    </form>
    <script nonce="INSERT_VALID_NONCE_VALUE">
        ...
    </script>
</div>
```

5.3.7 Tracking with custom domain

If your tracking domain is custom, then you need to define it with `img-src` and `script-src` directives:

```
img-src <your-sources> your-custom-cpp-domain.com;
script-src <your-sources> your-custom-cpp-domain.com;
```

5.3.8 Example Content Security Policy definition

Following example configuration of CSP assumes:

- client's website address: **client.com**
- Consent Manager is enabled for the website
- client's organization name in Piwik PRO: **client**
- client's container domain: **client.containers.piwik.pro**
- client has Piwik PRO tag with default tracking domain: **client.piwik.pro**
- nonce value: **nceIOfn39fn3e9h3sd**
- configuration allows `'self'` source which is: **client.com**

```
Content-Security-Policy: default-src 'self';
                         script-src  'self' client.piwik.pro 'nonce-nceIOfn39fn3e9h3sd
↪';
                         connect-src 'self' client.containers.piwik.pro client.piwik.
↪pro;
                         img-src     'self' client.containers.piwik.pro client.piwik.
↪pro;
                         font-src    'self' client.containers.piwik.pro;
                         style-src   'self' client.containers.piwik.pro 'nonce-
↪nceIOfn39fn3e9h3sd';
```

New in version 10.1.

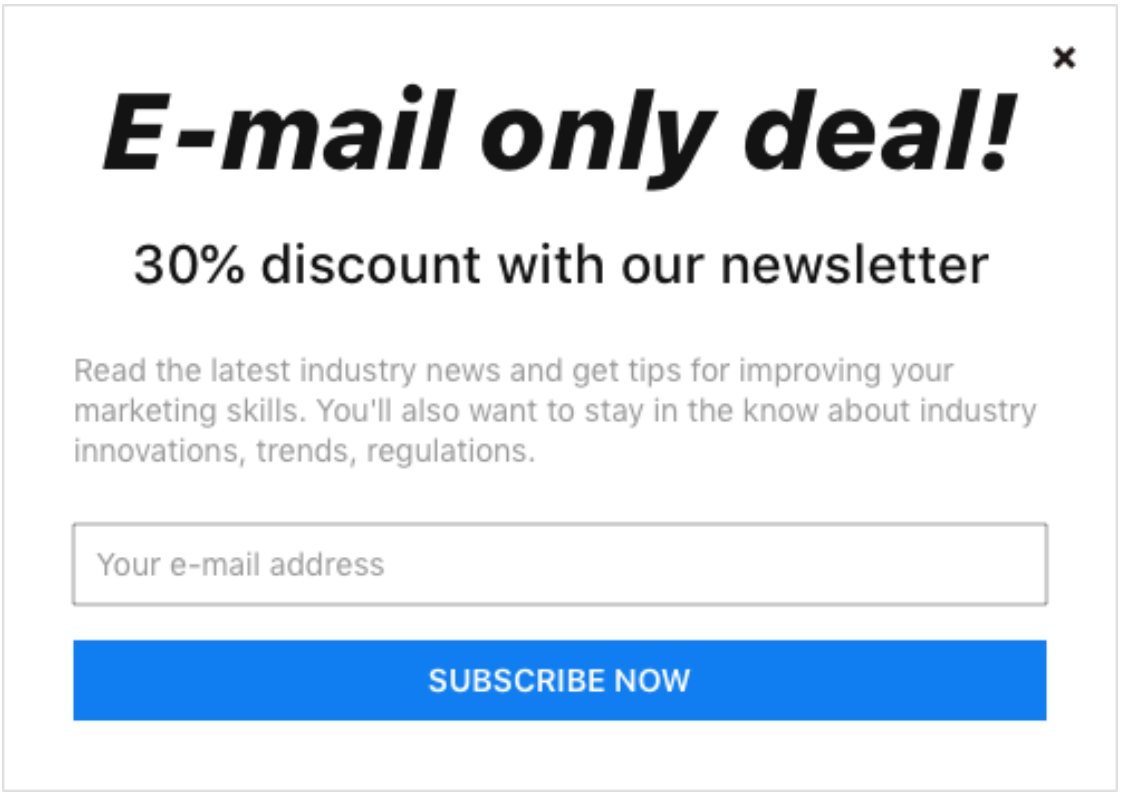
5.4 Custom popup template implementation examples

5.4.1 Introduction

Since version 10.1 of PPAS there is a possiblity of creating a *Custom popup* tag template. To add one, head to *Tag Manager* and while on *Tags* tab, choose + *Crate new tag*. From there you can select *Custom popup* template. Once added, you will be greated by default template code which consists of overlay, popup box and close button. To highlight what can be created with the use of this template, we decided to share some example implementations that can be further modified and expanded.

5.4.2 Example 1

Preview:



Note: Handling of the close button is provided out of the box, as long as the class name `ppms-popup-close-button` is unchanged. Your own JavaScript code to handle *Subscribe now* button needs to be provided.

Example code:

```
<div class="ppms-popup-overlay">
  <div class="ppms-popup-box">
    <span class="ppms-popup-close-button"> <!-- classname must stay as it is,
↪ otherwise close button will not work -->
      <svg width="16px" height="16px" viewBox="0 0 16 16" version="1.1"␣
↪xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/
↪xlink">
```

(continues on next page)

(continued from previous page)

```
        <g>
            <path d="M11.125,3 L13,4.875 L9.874,7.999 L13,11.125 L11.125,13
↪L7.999,9.874 L4.875,13 L3,11.125 L6.125,7.999 L3,4.87L4.875,3 L7.999,6.125
↪L11.125,3 Z" />
        </g>
      </svg>
    </span>
    <div class="ppms-popup-content">
      <h1 class="ppms-popup-header">E-mail only deal!</h1>
      <h2 class="ppms-popup-subheader">30% discount with our newsletter</h2>
      <p class="ppms-popup-paragraph">
        Read the latest industry news and get tips for improving your
↪marketing skills.
        You'll also want to stay in the know about industry innovations,
↪trends, regulations.
      </p>
      <input class="ppms-popup-input" type="email" placeholder="Your e-mail
↪address">
      <button class="ppms-popup-button">Subscribe now</button>
    </div>
  </div>
</div>


<style type="text/css">
  .ppms-popup-overlay {
    z-index: 10000;
    width: 100%;
    height: 100%;
    top: 0;
    left: 0;
    position: fixed;
    background-color: rgba(0, 0, 0, 0.8);
    display: flex;
    justify-content: center;
    align-items: center;
  }

  .ppms-popup-box {
    max-width: 500px;
    min-height: 350px;
    box-sizing: border-box;
    position: relative;
    background-color: #fff;
    border: 1px solid #ddd;
    padding: 28px 32px 32px 32px;
  }

  .ppms-popup-close-button {
    z-index: 1000;
    right: 16px;
    top: 16px;
    position: absolute;
    cursor: pointer;
    box-sizing: content-box;
    fill: #000;
  }
```

(continues on next page)

(continued from previous page)

```
  .ppms-popup-close-button:hover {
    fill: #999;
  }

  .ppms-popup-content {
    font-family: "BlinkMacSystemFont", -apple-system, "Roboto", "Oxygen-Sans
↪", "Ubuntu", "Cantarell", "Helvetica Neue", sans-serif;
  }

  .ppms-popup-header {
    text-align: center;
    font-style: italic;
    font-size: 48px;
    line-height: 58px;
    color: #131313;
    font-weight: 700;
    margin: 0;
  }

  .ppms-popup-subheader {
    color: #131313;
    font-size: 24px;
    font-weight: 500;
    line-height: 29px;
    text-align: center;
    margin-top: 16px;
  }

  .ppms-popup-paragraph {
    color: #999999;
    font-size: 14px;
    line-height: 18px;
    margin-top: 24px;
  }

  .ppms-popup-input {
    display: block;
    width: 100%;
    box-sizing: border-box;
    height: 36px;
    border: 1px solid #999999;
    background-color: #FFFFFF;
    color: #999999;
    font-size: 14px;
    line-height: 16px;
    margin-top: 24px;
    padding: 0 10px;
  }

  .ppms-popup-input::placeholder {
    color: #999999;
  }

  .ppms-popup-button {
    height: 36px;
    background-color: #107ef1;
```

(continues on next page)

(continued from previous page)

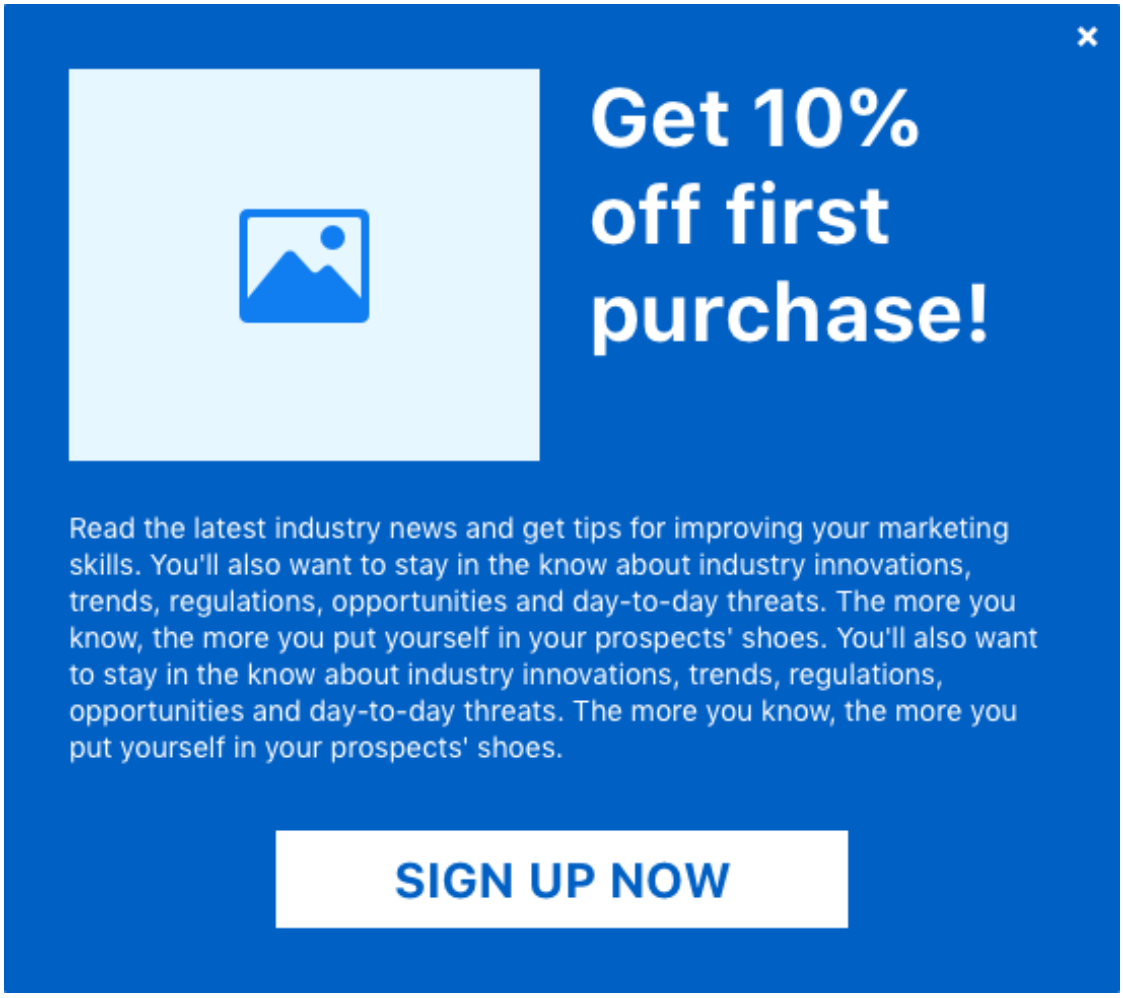
```
    color: #ffffff;
    width: 100%;
    text-transform: uppercase;
    border: none;
    font-size: 14px;
    font-weight: 600;
    line-height: 16px;
    text-align: center;
    margin-top: 16px;
    cursor: pointer;
  }

  .ppms-popup-button:hover {
    background-color: #338dee;
  }

  @media (max-height: 360px) {
    .ppms-popup-box {
      padding: 20px;
      min-height: unset;
    }
  }
</style>
```

5.4.3 Example 2

Preview:



Note: Handling of the close button is provided out of the box, as long as the class name `ppms-popup-close-button` is unchanged. Your own JavaScript code to handle *Sign up now* button needs to be provided.

Example code:

```
<div class="ppms-popup-overlay">
  <div class="ppms-popup-box">
    <span class="ppms-popup-close-button"> <!-- classname must stay as it is,
↪ otherwise close button will not work -->
      <svg width="16px" height="16px" viewBox="0 0 16 16" version="1.1"␣
↪xmlns="http://www.w3.org/2000/svg"
        xmlns:xlink="http://www.w3.org/1999/xlink">
        <g>
          <path d="M11.125,3 L13,4.875 L9.874,7.999 L13,11.125 L11.125,13 L7.
↪999,9.874 L4.875,13 L3,11.125 L6.125,7.999 L3,4.87L4.875,3 L7.999,6.125␣
↪L11.125,3 Z" />
        </g>
      </svg>
    </span>
    <!-- classname must stay as it is, otherwise close button will not work -
↪->
```

(continues on next page)

(continued from previous page)

```
    <div class="ppms-popup-content">
      <div class="ppms-popup-top-wrapper">
        <div class="ppms-popup-image">
          <svg width="64px" height="56px" viewBox="0 0 64 56" version="1.1"␣
↪xmlns="http://www.w3.org/2000/svg"
           xmlns:xlink="http://www.w3.org/1999/xlink">
           <g transform="translate(-869.000000, -538.000000)">
             <g transform="translate(48.000000, 538.000000)">
               <path d="M871.25,18.25 C870.083328,19.416672 868.666672,20␣
↪867,20 C865.333328,20 863.916672,19.416672
                 862.75,18.25 C861.583328,17.083328 861,15.666672 861,14␣
↪C861,12.333328 861.583328,10.916672 862.75,9.75
                 C863.916672,8.583328 865.333328,8 867,8 C868.666672,8 870.
↪083328,8.583328 871.25,9.75 C872.416672,
                 10.916672 873,12.333328 873,14 C873,15.666672 872.416672,
↪17.083328 871.25,18.25 Z M881,0 C882.142866,0
                 883.095232,0.388882667 883.857143,1.16666667 C884.619054,1.
↪94445067 885,2.91665733 885,4.08333333 L885,
                 51.9166667 C885,53.0833389 884.619054,54.0555521 883.
↪857143,54.8333333 C883.095232,55.611115 882.142866,
                 56 881,56 L825,56 C823.857137,56 822.904765,55.611115 822.
↪142857,54.8333333 C821.380949,54.0555521 821,
                 53.0833389 821,51.9166667 L821,4.08333333 C821,2.91665733␣
↪821.380949,1.94445067 822.142857,1.16666667
                 C822.904765,0.388882667 823.857137,0 825,0 L881,0 Z M866.
↪5625,28.4117647 L881,44 L881,5.76470588 C881,
                 4.58822588 880.368059,4 879.104167,4 L826.895833,4 C825.
↪826384,4 825.194445,4.58822588 825,5.76470588
                 L825,44 L843.375,21.6470588 C844.152784,20.8627388 844.
↪979167,20.4705882 845.854167,20.4705882
                 C846.923617,20.4705882 847.75,20.8137224 848.333333,21.5␣
↪L856.208333,30.1764706 L856.791667,30.7647059
                 C857.375,31.1568659 857.909716,31.3529412 858.395833,31.
↪3529412 C858.881951,31.3529412 859.465275,
                 31.1078494 860.145833,30.6176471 L862.770833,28.2647059␣
↪C863.451392,27.7745035 864.083333,27.5294118
                 864.666667,27.5294118 C865.444451,27.5294118 866.076383,27.
↪8235294 866.5625,28.4117647 Z" />
             </g>
           </g>
         </svg>
       </div>
       <h1 class="ppms-popup-header">Get 10% off first purchase!</h1>
     </div>
     <p class="ppms-popup-paragraph">
       Read the latest industry news and get tips for improving your␣
↪marketing skills.
       You'll also want to stay in the know about industry innovations,␣
↪trends, regulations, opportunities and
       day-to-day threats. The more you know, the more you put yourself in␣
↪your prospects' shoes. You'll also want to
       stay in the know about industry innovations, trends, regulations,␣
↪opportunities and day-to-day threats. The more
       you know, the more you put yourself in your prospects' shoes.
     </p>
     <button class="ppms-popup-button">Sign up now</button>
   </div>
```

(continues on next page)

(continued from previous page)

```
  </div>
</div>


<style type="text/css">
  .ppms-popup-overlay {
    z-index: 10000;
    width: 100%;
    height: 100%;
    top: 0;
    left: 0;
    position: fixed;
    background-color: rgba(0, 0, 0, 0.8);
    display: flex;
    justify-content: center;
    align-items: center;
  }

  .ppms-popup-box {
    width: 550px;
    min-height: 487px;
    box-sizing: border-box;
    position: relative;
    background-color: #0060c4;
    padding: 32px;
  }

  .ppms-popup-close-button {
    z-index: 1000;
    right: 8px;
    top: 8px;
    position: absolute;
    cursor: pointer;
    box-sizing: content-box;
    fill: #fff;
  }

  .ppms-popup-close-button:hover {
    fill: #aaa;
  }

  .ppms-popup-content {
    font-family: "BlinkMacSystemFont", -apple-system, "Roboto", "Oxygen-Sans
↪", "Ubuntu", "Cantarell", "Helvetica Neue", sans-serif;
  }

  .ppms-popup-top-wrapper {
    display: flex;
    flex-wrap: wrap;
    align-items: top;
    margin: -12px;
  }

  .ppms-popup-image {
    flex: 1 1 232px;
    height: 193px;
    background-color: #e6f7ff;
```

(continues on next page)

(continued from previous page)

```
  display: flex;
  justify-content: center;
  align-items: center;
  margin: 12px;
  fill: #107EF1;
}

.ppms-popup-header {
  flex: 1 1 230px;
  text-align: left;
  color: #fff;
  font-size: 40px;
  font-weight: bold;
  line-height: 48px;
  margin: 12px;
}

.ppms-popup-paragraph {
  color: #fff;
  font-size: 14px;
  line-height: 18px;
  margin-top: 24px;
}

.ppms-popup-button {
  display: block;
  width: 282px;
  height: 48px;
  background-color: #fff;
  color: #0060C4;
  font-size: 24px;
  font-weight: bold;
  line-height: 29px;
  text-align: center;
  text-transform: uppercase;
  border: none;
  margin: 32px auto 0 auto;
  cursor: pointer;
}

.ppms-popup-button:hover {
  background-color: #aaa;
}

@media (max-width: 560px) {
  .ppms-popup-image {
    display: none;
  }

  .ppms-popup-box {
    display: flex;
    align-items: center;
  }

  .ppms-popup-button {
    padding: 0 25px;
    width: auto;
```

(continues on next page)

(continued from previous page)

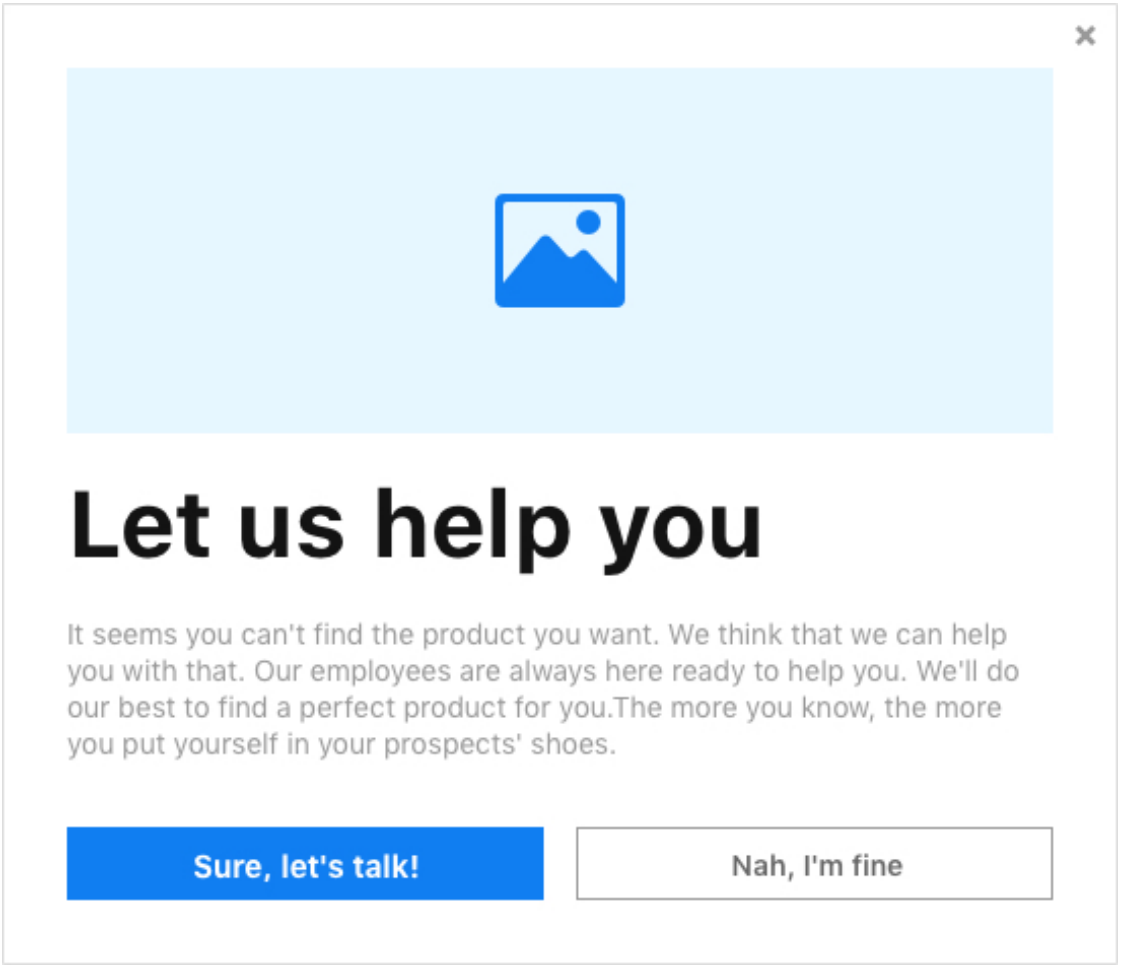
```
    }
  }

  @media (max-height: 490px) {
    .ppms-popup-image {
      display: none;
    }

    .ppms-popup-box {
      width: 100%;
      display: flex;
      align-items: center;
      min-height: unset;
      padding: 20px;
    }
  }
</style>
```

5.4.4 Example 3

Preview:



Note: Handling of the close button is provided out of the box, as long as the class name `ppms-popup-close-button` is unchanged. Your own JavaScript code to handle *Sure, let's talk* and *Nah, I'm fine* buttons needs to be provided.

Example code:

```
<div class="ppms-popup-overlay">
  <div class="ppms-popup-box">
    <span class="ppms-popup-close-button"> <!-- classname must stay as it is,
↪ otherwise close button will not work -->
      <svg width="16px" height="16px" viewBox="0 0 16 16" version="1.1"␣
↪xmlns="http://www.w3.org/2000/svg"
       xmlns:xlink="http://www.w3.org/1999/xlink">
        <g>
          <path d="M11.125,3 L13,4.875 L9.874,7.999 L13,11.125 L11.125,13 L7.
↪999,9.874 L4.875,13 L3,11.125 L6.125,7.999 L3,4.87L4.875,3 L7.999,6.125␣
↪L11.125,3 Z" />
        </g>
      </svg>
    </span>
    <div class="ppms-popup-content">
      <div class="ppms-popup-image">
        <svg width="64px" height="56px" viewBox="0 0 64 56" version="1.1"␣
↪xmlns="http://www.w3.org/2000/svg"
         xmlns:xlink="http://www.w3.org/1999/xlink">
          <g transform="translate(-869.000000, -538.000000)">
            <g transform="translate(48.000000, 538.000000)">
              <path d="M871.25,18.25 C870.083328,19.416672 868.666672,20 867,
↪20 C865.333328,20 863.916672,19.416672
                862.75,18.25 C861.583328,17.083328 861,15.666672 861,14 C861,
↪12.333328 861.583328,10.916672 862.75,9.75
                C863.916672,8.583328 865.333328,8 867,8 C868.666672,8 870.
↪083328,8.583328 871.25,9.75 C872.416672,
                10.916672 873,12.333328 873,14 C873,15.666672 872.416672,17.
↪083328 871.25,18.25 Z M881,0 C882.142866,0
                883.095232,0.388882667 883.857143,1.16666667 C884.619054,1.
↪94445067 885,2.91665733 885,4.08333333 L885,
                51.9166667 C885,53.0833389 884.619054,54.0555521 883.857143,
↪54.8333333 C883.095232,55.611115 882.142866,
                56 881,56 L825,56 C823.857137,56 822.904765,55.611115 822.
↪142857,54.8333333 C821.380949,54.0555521 821,
                53.0833389 821,51.9166667 L821,4.08333333 C821,2.91665733␣
↪821.380949,1.94445067 822.142857,1.16666667
                C822.904765,0.388882667 823.857137,0 825,0 L881,0 Z M866.
↪5625,28.4117647 L881,44 L881,5.76470588 C881,
                4.58822588 880.368059,4 879.104167,4 L826.895833,4 C825.
↪826384,4 825.194445,4.58822588 825,5.76470588
                L825,44 L843.375,21.6470588 C844.152784,20.8627388 844.
↪979167,20.4705882 845.854167,20.4705882
                C846.923617,20.4705882 847.75,20.8137224 848.333333,21.5␣
↪L856.208333,30.1764706 L856.791667,30.7647059
                C857.375,31.1568659 857.909716,31.3529412 858.395833,31.
↪3529412 C858.881951,31.3529412 859.465275,
                31.1078494 860.145833,30.6176471 L862.770833,28.2647059 C863.
↪451392,27.7745035 864.083333,27.5294118
                864.666667,27.5294118 C865.444451,27.5294118 866.076383,27.
↪8235294 866.5625,28.4117647 Z" />
```

(continues on next page)

(continued from previous page)

```
            </g>
          </g>
        </svg>
      </div>
      <h1 class="ppms-popup-header">Let us help you</h1>
      <p class="ppms-popup-paragraph">
        It seems you can't find the product you want. We think that we can␣
↪help you with that. Our employees are always
        here ready to help you. We'll do our best to find a perfect product␣
↪for you.The more you know, the more you put
        yourself in your prospects' shoes.
      </p>
      <div class="ppms-popup-button-wrapper">
        <button class="ppms-popup-button ppms-popup-button-accept">Sure, let
↪'s talk!</button>
        <button class="ppms-popup-button ppms-popup-button-reject">Nah, I'm␣
↪fine</button>
      </div>
    </div>
  </div>
</div>


<style type="text/css">
  .ppms-popup-overlay {
    z-index: 10000;
    width: 100%;
    height: 100%;
    top: 0;
    left: 0;
    position: fixed;
    background-color: rgba(0, 0, 0, 0.8);
    display: flex;
    justify-content: center;
    align-items: center;
  }

  .ppms-popup-box {
    width: 550px;
    min-height: 474px;
    box-sizing: border-box;
    position: relative;
    background-color: #fff;
    padding: 32px;
  }

  .ppms-popup-close-button {
    z-index: 1000;
    right: 8px;
    top: 8px;
    position: absolute;
    cursor: pointer;
    box-sizing: content-box;
    fill: #999;
  }

  .ppms-popup-close-button:hover {
```

(continues on next page)

(continued from previous page)

```
    fill: #666;
  }

  .ppms-popup-content {
    font-family: "BlinkMacSystemFont", -apple-system, "Roboto", "Oxygen-Sans
↪", "Ubuntu", "Cantarell", "Helvetica Neue", sans-serif;
  }

  .ppms-popup-image {
    width: 100%;
    height: 180px;
    background-color: #e6f7ff;
    display: flex;
    justify-content: center;
    align-items: center;
    fill: #107EF1;
  }

  .ppms-popup-header {
    text-align: left;
    color: #000;
    font-size: 46px;
    font-weight: bold;
    margin: 16px 0;
  }

  .ppms-popup-paragraph {
    color: #999;
    font-size: 14px;
    line-height: 18px;
    margin-bottom: 32px;
  }

  .ppms-popup-button-wrapper {
    display: flex;
    flex-wrap: wrap;
    margin: -8px;
  }

  .ppms-popup-button {
    height: 36px;
    flex: 1 1 235px;
    font-size: 15px;
    font-weight: bold;
    line-height: 18px;
    text-align: center;
    padding: 0px;
    margin: 8px;
    cursor: pointer;
  }

  .ppms-popup-button-accept {
    background-color: #1c80eb;
    color: #fff;
    border: none;
  }
```

(continues on next page)

(continued from previous page)

```
  .ppms-popup-button-accept:hover {
    background-color: #338dee;
  }

  .ppms-popup-button-reject {
    background-color: #fff;
    color: #666;
    border: 1px solid #999;
  }

  .ppms-popup-button-reject:hover {
    background-color: #eee;
  }

  @media (max-width: 560px), (max-height: 480px) {
    .ppms-popup-image {
      display: none;
    }

    .ppms-popup-box {
      display: flex;
      align-items: center;
      min-height: unset;
    }
  }
</style>
```

New in version 14.0.

5.5 Skip link tracking with *data-disable-delay* attribute

5.5.1 Introduction

As per the MDN definition:

> The <a> HTML element (or anchor element), with its href attribute, creates a hyperlink to web pages, files, email addresses, locations in the same page, or anything else a URL can address.

If you wish to trigger tags, when the anchor element is clicked, they need time to execute before the redirect happens. That is why our container is equipped with a delay mechanism.

5.5.2 Delay mechanism

Each app or site you create has the option to *Delay loading the next page* in its settings. You can adjust this value in *Data collection -> Other options* section in your app settings under *Administration -> Sites & apps*. The default value for each app is `500ms`. Once you assign a trigger and a tag to an anchor element on your page, this mechanism will ensure that the tag fires and has time to execute, before the visitor is redirected to the desired page.

However, not every anchor element is supposed to perform a redirect. One such example can be SPA pages where `a` elements can serve as buttons. In such case, the action performed inside the container can break the functionality of the page. That is where the `data-disable-delay` attribute comes in.

5.5.3 *data-disable-delay* attribute

`data-disable-dalay` is special custom attribute that is recognized by the container. Once the anchor element is clicked and the aforementioned attribute is detected on the element, it tells the container to skip the execution of the logic responsible for delaying the default action. Listeners attached to the element are executed immediatly after clicking.

Example

1. Let's assume that your Tag Manager setup includes a *Custom code (async)* tag (the contents of the tag does not matter in this case) and a basic *Click trigger* assigned to the said tag.
2. On your page, the following code is present:

```
<a
    id='link-id'
    href="/"
>
    link
</a>
<script>
    window.setTimeout(function() {
        document.addEventListener('click', function(event) {
            if(event.target.id === 'link-id') {
                event.preventDefault()
            }
        })
    }, 1000)
</script>
```

3. Once the visitor clicks the link, a redirect happens. This is not desired, since the listener performs a *preventDefault* action.
4. Now let's modify the anchor element to look like this:

```
<a
    id='link-id'
    href="/"
    data-disable-delay
>
    link
</a>
```

5. After the modification is done, clicking the link no longer performs a redirect and fires the click listener immediately.

CHAPTER 6

Administration

6.1 Getting started

6.1.1 Create API credentials and an access token

If you want to access API for the first time, you need to generate your API credentials and use them to create an access token. The token is needed to authenticate API calls.

Our API uses client credentials (OAuth grant type) for obtaining a user token. All data is sent and received as JSON and is compliant with the JSON API specification.

Generate API credentials

To generate API credentials, follow these steps:

1. Log in to Piwik PRO.
2. Go to Menu > Profile.
3. Navigate to API credentials.
4. Click Generate new credentials.
5. Enter Name and click OK.
6. Copy Client ID and Client secret. They won't be available after you close this window.

Note: Credentials are valid until they are deleted in the Profile.

Create an access token

To create an access token, follow these steps:

1. Piwik PRO API tokens use JWT format.
2. Make a call:

```
curl -X POST 'https://<example>/auth/token' -H "Content-Type: application/json" --
↪data '{
    "grant_type": "client_credentials",
    "client_id": "<client_id>",
    "client_secret": "<client_secret>"
}'
```

Note: If you are the Core plan user, replace <example> with <your_account_name>.piwik.pro.

3. Response example:

```
{"token_type":"Bearer","expires_in":1800,"access_token":"<your_access_token>"}
```

4. Now you can use <your_access_token> to communicate with Piwik PRO API. The token is a Bearer type, so you need to include it within the header in every API call.

```
Authorization: Bearer <your_access_token>
```

Note: Every token is valid for 30 minutes. expires_in shows the expiration time in seconds.

Delete API credentials

If you no longer want to use generated API credentials in access tokens, you need to delete them.

To delete API credentials, follow these steps:

1. Log in to Piwik PRO.
2. Go to Menu > Profile.
3. Navigate to API credentials.
4. Choose credentials that you want to revoke and click X.

6.1.2 Examples of using API

Note: To use any API call, you need to have API credentials (see above).

Using API with curl

In this example, we want to perform some basic operations on a user. We'll do the following operations:

- Invite a user
- Get a created user
- Change the user's language
- Delete a user

Note: In our example, we use https://<example> as an account address. An account address has this format: https://example.piwik.pro.

Generate your access token

Example of a request:

POST /auth/token

```
curl -X POST 'https://<example>/auth/token' -H "Content-Type: application/json" --
↪data '{
    "grant_type": "client_credentials",
    "client_id": "your_generated_client_id",
    "client_secret": "your_generated_client_secret"
}'
```

Response example:

```
{
    "token_type":"Bearer",
    "expires_in":1800,
    "access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.
↪eyJpc3MiOiJwcG1zIiwiYXVkIjoiaHR0cHM6XC9cL3Rlc3RpbmcucGl3aWsucHJvXC9zZXR5LCJzdWIiOiJkNmNkZGMxMS1iZDA
↪Nec2mYFRv6manzXjq0sHQxINZvu-fbDYT8AedVHBKYvu1F9hYKaFReY8rNgfsMANw2OX8-
↪IKpTrQb1DyRkG4nxpIEbob528_
↪lPd7roho5mtKlE8sfS9WZE1piYOwaNDySDEUwUowgj2xBiJqSODjxBI6qVhLkynGEEeNBVh-
↪lrUmlcjpYqUc3saHvX72L-rqbIHa_1dzGarR-dcPyns-RpKjZEILzUSYOHdM09KDti-xsG-
↪nbKHGdP8fVEEJPyupnAfJPOLHQg_j1c5IvJSvTKVF3j4_
↪zo6Zw5g8YkaheT9Iwph5BGHFRneXatcmbwKI8JzSDFi6CinzI-okYKRPbg"
}
```

Note: access_token contains your token. You'll need it for all API calls. Every token is valid for 30 minutes.

Invite a user

Request example:

POST /api/users/v2

```
curl -X POST 'https://<example>/api/users/v2' -H "Authorization: Bearer <your_access_
↪token>" -H "Content-Type: application/vnd.api+json" --data '{
  "data": {
    "type": "ppms/user",
    "attributes": {
      "email": "user@example.com",
      "language": "en-US"
    }
  }
}'
```

Replace in your request the following fields:

- <example> with your account address. Example: example.piwik.pro.
- <your_access_token> with your generated access token

Example of a response:

```
{
  "data": {
    "id": "b30e538d-4b05-4a75-ae25-7eb565901f38",
```

(continues on next page)

(continued from previous page)

```
    "type": "ppms/user",
    "attributes": {
      "email": "user@example.com",
      "role": "USER",
      "addedAt": "2021-08-02T12:16:30+00:00",
      "language": "en-US"
    }
  }
}
```

Get a user

After inviting a user, you can get a user.

Request example:

GET /api/users/v2/<user_id>

```
curl 'https://<example>/api/users/v2/b30e538d-4b05-4a75-ae25-7eb565901f38' -H
↪"Authorization: Bearer <your_access_token>"
```

Note: The URL contains b30e538d-4b05-4a75-ae25-7eb565901f38. What is it? It is a user ID. If you want to update a given resource, you need to specify which one. You'll find a user ID in the `data/id` field in the response for adding a user.

Response example:

```
{
  "data": {
    "id": "b30e538d-4b05-4a75-ae25-7eb565901f38",
    "type": "ppms/user",
    "attributes": {
      "email": "user@example.com",
      "role": "USER",
      "addedAt": "2021-08-02T12:16:30+00:00",
      "language": "en-US"
    }
  }
}
```

Change the user's language

If you want to change the user's language after adding a user, you can use the following method.

Request example:

PATCH /api/users/v2/<user_id>

```
curl -X PATCH 'https://<example>/api/users/v2/b30e538d-4b05-4a75-ae25-7eb565901f38' -
↪H "Authorization: Bearer <your_access_token>" -H "Content-Type: application/vnd.
↪api+json" -v --data '{
  "data": {
    "type": "ppms/user",
    "id": "b30e538d-4b05-4a75-ae25-7eb565901f38",
```

(continues on next page)

(continued from previous page)

```
    "attributes": {
      "language": "de-DE"
    }
  }
}'
```

This request changed the user's language name from en-US to de-DE.

Here are some things to know:

- We use -X PATCH before the URL. It means that this request is available using HTTP PATCH method.
- You also need to specify data/id. It's a JSON API requirement.
- data/type is required. For example, when you want to work with a user resource, specify its type as ppms/user.
- You can set only parameters you want to update. For more user attributes, go to User edit reference

API will return 204 No Content status code with an empty response.

Delete a user

When you want to remove a user, you can use the following method.

Request example:

DELETE /api/users/v2/<user_id>

```
curl -X DELETE 'https://<example>/api/users/v2/b30e538d-4b05-4a75-ae25-7eb565901f38' -
↪H "Authorization: Bearer <your_access_token>"
```

API will only return 204 No Content status code.

6.1.3 Using API with Postman

Postman is a multiplatform GUI application for creating API calls. Piwik PRO allows you to export Swagger documentation and easily import it to Postman. Depending of what you want to work with, you can import the following swagger docs:

- Access control
- Apps
- Audit log
- Meta Sites
- Modules
- Tracker settings
- Users
- User Groups

To use Postman, follow these steps:

1. In Postman, click import -> Import From Link.
2. Done. All of your paths are imported.
3. Now override two elements:

- Replace your domain in the URL.
- Add your token: In the selected API call, click Authorization. Use the Bearer Token type. Paste your token. Click SEND to call API.

6.1.4 FAQ

API returns "application/json" is not a valid JSON API Content-Type header, use "application/vnd.api+json" instead"

All API calls need to be created with the Content-Type: application/vnd.api+json header. If you use curl, you need to use the -H "Content-Type: application/vnd.api+json" flag. Postman allows configuring headers with the Header tab.

API returns JWT not found

You need to use your API token with every API call. Always send your API token within the Authorization: Bearer <your_access_token> header. If you use curl, you need to use the -H "Authorization: Bearer <your_access_token>" flag. Postman allows configuring tokens in the authorization tab. Choose the Bearer Token type and paste the token there. Remember to keep your token secure because it gives access to sensitive data.

API returns Expired JWT Token

Every token is valid for 30 minutes. After the token expires, you can create it again.

API returns access token not authorized

This message means that you sent an access token within a correct Authorization: Bearer field, but the token is invalid. Check your token and try again.

6.2 Access Control API

6.3 Apps API

6.4 Audit log API

6.5 Meta Sites API

6.6 Modules API

6.7 Collecting & Processing Pipeline Settings API

6.8 User Groups API

6.9 Users API

CHAPTER 7

Glossary

Analytics PPAS component gathering statistics about each *visitor* of the *application* (previously **Piwik**).

Analytics attribute *Attribute* generated from value provided by *Analytics* (e.g. browser and device data, location data, etc.). You can read more about *attribute* sources here.

> **Note:** If *custom attribute* uses the same name - it will be represented as a separate *attribute*.

Analytics ID ID assigned to *visitor* by *Analytics* for the duration of *Analytics* session. It is stored in browser cookie.

Application Website or application tracked by PPAS.

App ID PPAS *application* identificator (previously **website ID**, **site ID** or **idSite**).

Attribute Named value assigned to *visitor* profile.

Attribute whitelist List of *visitor attributes* that are publicly available via Audience Manager API.

> **Note:** It is still necessary to identify the *visitor* with his *analytics ID* to access this information.

Audience Named set of *attribute* conditions used to define a group of visitors matching them.

Custom attribute *Attribute* generated from value provided by source other than *Analytics* (e.g. *Form Tracker*, sdk/index). You can read more about *attribute* sources here.

> **Warning:** *Custom attribute* will store only latest value provided by any custom source.

> **Note:** If *analytics attribute* uses the same name - it will be represented as a separate *attribute*.

Collecting & Processing Pipeline Formerly called tracker backend. A set of services that is able to receive, store and process requests from JavaScript Tracking Client. Requests processed by Collecting & Processing Pipeline are passed further for Reporting services.

Device ID Device ID (device identification) is a distinctive number associated with a smartphone or similar handheld device. Device IDs are separate from hardware serial numbers.

Identifier Unique identifier of a *visitor* ID (e.g. *analytics ID*, *user ID*, *device ID* or email).

JavaScript Tracking Client (JSTC) A JavaScript object that is able to send requests to *Collecting & Processing Pipeline*. It is loaded and created with download of *ppms.js* file. It has an *API* that allows to configure what data requests should contain. You can *learn more about JSTC here*

JavaScript Tracking Snippet (JSTS) A JavaScript code, usually in form of HTML tag, that initiates JSTC and sends first tracking request. You can see an *example of JSTS here*.

PII Personally Identifiable Information.

Tracking Tag A HTML tag, that is created and configured by Tag Manager. It is loaded to the website with Tag Manager Container. Using a Tracking Tag is an alternative for implementing a JavaScript Tracking Snippet. You can learn more about Tracking Tag here.

User ID Permanent ID assigned to *visitor* by *application* (e.g. username). You can read more about it here.

Visit Period of continuous *visitor* activity on *application*. It ends in the following situations:

- after a period of inactivity (option set to 30 minutes by default)
- on campaign change (option enabled by default)
- when HTTP referrer points to different website (option disabled by default)

Visitor Visitor on tracked *application*.

Index

Symbols

_paq.push() (*_paq method*), 39

A

addDownloadExtensions() (*built-in function*), 57
addEcommerceItem() (*built-in function*), 43
addListener() (*built-in function*), 67
Analytics, **151**
Analytics attribute, **151**
Analytics ID, **151**
App ID, **151**
Application, **151**
Attribute, **151**
Attribute whitelist, **151**
Audience, **151**

C

clearEcommerceCart() (*built-in function*), 44
Collecting & Processing Pipeline, **152**
Custom attribute, **151**
customCrossDomainLinkDecorator() (*built-in function*), 62
customCrossDomainLinkVisitorIdGetter() (*built-in function*), 63

D

deanonymizeUser() (*built-in function*), 59
deleteCookies() (*built-in function*), 60
deleteCustomDimension() (*built-in function*), 49
deleteCustomVariable() (*built-in function*), 47
Device ID, **152**
device_id (*None attribute*), 108
disableCookies() (*built-in function*), 60
disableCrossDomainLinking() (*built-in function*), 61

E

email (*None attribute*), 108
enableCookies() (*built-in function*), 60
enableCrossDomainLinking() (*built-in function*), 61
enableHeartBeatTimer() (*built-in function*), 65
enableJSErrorTracking() (*built-in function*), 68
enableLinkTracking() (*built-in function*), 54

F

fields (*None attribute*), 110

G

getCrossDomainLinkingUrlParameter() (*built-in function*), 62
getCustomDimension() (*built-in function*), 50
getCustomDimensionValue() (*built-in function*), 49
getCustomVariable() (*built-in function*), 48
getEcommerceItems() (*built-in function*), 45
getLinkTrackingTimer() (*built-in function*), 66
getTimingDataSamplingOnPageLoad() (*built-in function*), 65
getVisitorId() (*built-in function*), 59
getVisitorInfo() (*built-in function*), 59

H

hasCookies() (*built-in function*), 60

I

Identifier, **152**
isCrossDomainLinkingEnabled() (*built-in function*), 61

J

JavaScript Tracking Client (*JSTC*), **152**
JavaScript Tracking Snippet (*JSTS*), **152**

L

logAllContentBlocksOnPage() (*built-in function*), 53

O

onFulfilled() (*built-in function*), 104, 105, 108, 114–120
onRejected() (*built-in function*), 104–106, 108, 115–120

P

PII, **152**
ping() (*built-in function*), 67
Piwik.getAsyncTracker() (*Piwik method*), 40
Piwik.getTracker() (*Piwik method*), 40
ppms.am.api() (*ppms.am method*), 103
ppms.cm.api() (*ppms.cm method*), 114

R

removeDownloadExtensions() (*built-in function*), 57
removeEcommerceItem() (*built-in function*), 44
resetUserId() (*built-in function*), 58

S

setCookieDomain() (*built-in function*), 60
setCookieNamePrefix() (*built-in function*), 60
setCookiePath() (*built-in function*), 60
setCrossDomainLinkingTimeout() (*built-in function*), 61
setCustomDimension() (*built-in function*), 50
setCustomDimensionValue() (*built-in function*), 48
setCustomRequestProcessing() (*built-in function*), 68
setCustomVariable() (*built-in function*), 47
setDocumentTitle() (*built-in function*), 64
setDomains() (*built-in function*), 63
setDownloadClasses() (*built-in function*), 56
setDownloadExtensions() (*built-in function*), 56
setEcommerceView() (*built-in function*), 45
setIgnoreClasses() (*built-in function*), 55
setLinkClasses() (*built-in function*), 55
setLinkTrackingTimer() (*built-in function*), 65
setReferralCookieTimeout() (*built-in function*), 61
setRequestContentType() (*built-in function*), 67
setRequestMethod() (*built-in function*), 67
setSecureCookie() (*built-in function*), 61
setSessionCookieTimeout() (*built-in function*), 61
setSiteInspectorSetup() (*built-in function*), 66
setTimingDataSamplingOnPageLoad() (*built-in function*), 64
setUserId() (*built-in function*), 58
setUserIsAnonymous() (*built-in function*), 58
setVisitorCookieTimeout() (*built-in function*), 61
setVisitorIdCookie() (*built-in function*), 61
storeCustomVariablesInCookie() (*built-in function*), 48

T

trackAllContentImpressions() (*built-in function*), 51
trackContentImpression() (*built-in function*), 52
trackContentImpressionsWithinNode() (*built-in function*), 52
trackContentInteraction() (*built-in function*), 53
trackContentInteractionNode() (*built-in function*), 53
trackEcommerceCartUpdate() (*built-in function*), 46
trackEcommerceOrder() (*built-in function*), 46
trackEvent() (*built-in function*), 41
trackGoal() (*built-in function*), 42
Tracking Tag, **152**
trackingType (*None attribute*), 111
trackLink() (*built-in function*), 54
trackPageView() (*built-in function*), 41
trackSiteSearch() (*built-in function*), 42
trackVisibleContentImpressions() (*built-in function*), 51
type (*None attribute*), 110

U

urlDecorator() (*built-in function*), 62
urlParser() (*built-in function*), 63
useLabels (*None attribute*), 111
User ID, **152**
user_id (*None attribute*), 108

V

Visit, **152**
Visitor, **152**