Piwik PRO Marketing Suite Documentation

Release 6.2

Piwik PRO

Sep 09, 2020

Contents

PPAS version: 6.3 (see changelog)

CHAPTER 1

SDK

1.1 Piwik PRO SDK for Android

1.1.1 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 "Settings > Websites" and your server address.

Client

Including the library

Add dependencies to your app module `build.gradle` file (e.g. `~/git/MyApplication/app/build.gradle`):

```
dependencies {
    repositories {
        jcenter()
    }
    compile 'pro.piwik.sdk:piwik-sdk:VERSION'
}
```

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

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 "Settings > Websites"):

```
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.

1.1.2 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.

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;
            }

            @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));
```

(continues on next page)

(continued from previous page)

```
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?_rcn=Email-
↪SummerDeals&_rck=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);
```

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.

1.1.3 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 single period of user interaction with your app. By default, Piwik will group hits that are received within 30 minutes of one another into the same session. You can configure Piwik to automatically start new sessions when users have placed your app in the background for a period of time. This session timeout 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);

      ...
    }
```

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.

1.1.4 License

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

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.

1.2 Piwik PRO SDK for iOS

1.2.1 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 "Settings > Websites" 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 *Settings* > *Websites* 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;
}
```

1.2.2 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.

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.

1.2.3 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. We recommend using userID instead of VisitorID.

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;
```

1.2.4 License

Piwik PRO iOS SDK is available under the MIT license.

CHAPTER 2

Analytics

2.1 JavaScript tracking 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

2.1.1 Installing Tracking code

There are two ways of installing a tracking code:

Installing tracking code via Tag Manager

This is the easiest and recommended way of tracking code installation. When Tag Manager is added to the site - it automatically publishes tracking code (using "Piwik PRO Analytics template").

If you do not have Tag Manager on your website yet, follow this procedure to install it: #. Sign in to your PPAS with your admin or Super User account. #. Click on the menu button on the top left. #. Click on the "Websites" position. #. Choose the website for which you want to implement a tracking code. #. Select the "Installation" tab. #. The Tag Manager code snippet for your website is displayed under the "Website code for asynchronous tags" or "Website code for synchronous tags".

Installing tracking code via code snippet.

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 multi-domain 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>
```

Deprecated since version 5.5.1: Older installations using `piwik.php` and `piwik.js` filenames are deprecated.

This code initializes the Analytics tracker in following ways:

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

When loading, the snippet is added on the page. The Analytics tracker will start tracking *user* actions starting with page view.

Alternative multi-domain 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"]);

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

Deprecated since version 5.5.1: Older installations using `piwik.php` and `piwik.js` filenames are deprecated.

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:

```
_paq.push(["setDomains", domains]);
```

`domains`
Required `array` Domains array, with wildcards

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

Tracking subdirectories of domain as separate websites

To differentiate parts of a website as another site, you must configure tracker 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"]);
```

In 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.

Deprecated since version 5.5.1: Older installations using `piwik.php` and `piwik.js` filenames are deprecated.

2.1.2 Command queue

Loading snippet creates the following API function:

_paq.**push**(*command*)
JavaScript API interface.

Arguments

- **command** (*Array<string>*) – Array containing command *name* followed by its arguments. The number of arguments and their function depend on command.

Return type undefined

2.1.3 Commands

Trigger tracking on demand

Trigger custom event

Trigger (custom) events bound to user actions:

```
_paq.push(["trackEvent", category, action, name, value, dimensions]);
```

category
Required `string` Event category.

action
Required `string` Event action.

name
Optional `string` Event name.

value
Optional `number` Event value.

dimensions
Optional `object` *Custom dimensions* which should be tracked using this action. It can set multiple dimensions at once. Dimensions are defined as object properties using the `dimension{ID}` notation.

Example:

```
{
   dimension1: "example value",
   dimension2: "example value"
}
```

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

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

Track goal conversion

Allows the manual tracking of goal conversion. Used in *Goals - Days to Conversion* report. Command:

```
_paq.push(["trackGoal", goal_name, goal_value, dimensions]);
```

goal_name
: **Required** `string` Goal Name

goal_value
: **Optional** `number` Tracked conversion value.

dimensions
: **Optional** `object` *Custom dimensions* which should be tracked using this action. Dimensions are defined as object properties using the `dimension{ID}` notation.

  Example:

```
{
   dimension1: "example value",
   dimension2: "example value"
}
```

Example of usage:

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

Ecommerce tracking

Adding Ecommerce item

To add an e-commerce item (for example to track changes in the user's cart using `trackEcommerceCartUpdate`), use the `addEcommerceItem` function:

```
_paq.push(["addEcommerceItem", productSKU, productName, productCategory, productPrice,
↪ productQuantity]);
```

Note: This function does not send any data to the *Analytics*. It only prepares E-commerce cart/order state to be sent with *trackEcommerceOrder* or *trackEcommerceCartUpdate*.

---

productSKU
: **Required** `string` Product stock-keeping unit.

productName
: **Optional** `string` Product name.

productCategory
: **Optional** `Array<string>|string` Product category, can be written as Array with up to 5 elements.

productPrice
: **Optional** `number` with product price.

productQuantity
: **Optional** `number` with product quantity.

Warning: Product SKU, names and categories should be URL encoded.

Warning: The state of the cart is not maintained across the visit. You must add all products after each page view.

Example of usage:

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

Tracking Ecommerce order

To successfully track the e-commerce order(s) (on the checkout page, for example) use the `trackEcommerceOrder` function:

```
_paq.push(["trackEcommerceOrder", orderId, orderGrandTotal, orderSubTotal, orderTax,␣
↪orderShipping, orderDiscount]);
```

`orderId`
Required `string` Unique order ID.

`orderGrandTotal`
Required `number` Order Revenue grand total - tax, shipping and discount included.

`orderSubTotal`
Optional `number` Order subtotal - without shipping.

`orderTax`
Optional `number` Order tax amount.

`orderShipping`
Optional `number` Order shipping costs.

`orderDiscount`
Optional `number` Order discount amount.

Example of usage:

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

Updating cart

To update the user cart (when the user adds new products or removes them from cart) use the `trackEcommerceCartUpdate` function:

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

`cartAmount`
Required `number` Cart amount (sum of products).

> **Warning:** Before updating the tracking cart, be sure to add all products in the cart by using `addEcommerceItem` first (including the ones that were previously in the cart). Then, use this function.

Example of usage:

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

Tracking product / category view

If you wish to track when the user enters the product site or is browsing products category, use the `setEcommerceView` function:

```
_paq.push(["setEcommerceView", productSKU, productName, productCategory,␣
↪productPrice]);
```

productSKU
: **Required** `string/boolean` Product stock-keeping unit. False for tracking category.

productName
: **Optional** `string/boolean` Product name. False for tracking category.

productCategory
: **Optional** `Array<string>|string` Product category, can be written as Array with up to 5 elements.

productPrice
: **Optional** `number` Product price.

> **Warning:** Product SKU, names and categories should be URL encoded.

Example of usage:

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

Custom Variables

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

Adding / Editing Custom Variable

To set a custom variable that can be used later, use the `setCustomVariable` function:

```
_paq.push(["setCustomVariable", index, name, value, scope]);
```

index
: **Required** `number` Index from 1 to 5 where the variable is stored

name
: **Required** `string` Name of the variable

value
: **Optional** `string` Value of the variable limited to 200 characters.

scope
: **Optional** `string` Scope of the variable, “visit” or “page”. The default value is `"visit"`

> **Note:** A Custom Variable with the scope set on “visit” will be saved for visit, you don’t need to save it for every page.

Warning: Index is separate for each variable scope.

Example of usage:

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

Removing Custom Variable

To remove the custom variable, you can use the `deleteCustomVariable` function:

```
_paq.push(["deleteCustomVariable", index, scope]);
```

index
Required `number` Index from 1 to 5 where the variable is stored

scope
Optional `string` Scope of the variable, "visit" or "page". The default value is `"visit"`

Example of usage:

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

Accessing Custom Variable

You can access custom variables by providing a function that will use the `getCustomVariable` function:

```
_paq.push([ function() {
    var customVariable = this.getCustomVariable(index, scope );
}]);
```

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

Arguments

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

Example of usage:

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

Custom Dimensions

Tracking Custom Dimension

If you wish to set a custom dimension to use it in tracking functions, use the `setCustomDimension` function:

```
_paq.push(["setCustomDimension", customDimensionID, customDimensionValue]);
```

customDimensionID
Required `number` ID of dimension

customDimensionValue
Required `string` Value of Custom Dimension - limited to 255 characters.

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 *Analytics*. It sets a Custom Dimension to be sent with following events (e.g. page view, ecommerce events, outlink or download events).

Example of usage:

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

Retrieving Custom Dimension

You can access custom dimension by providing a function that will use the `getCustomDimension` function:

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

getCustomDimension(*index*)

Arguments

- **index** (*number*) – **Required** Index of custom dimension

Example of usage:

```
_paq.push([ function() {
    var customDimension = this.getCustomDimension(1);
    console.log(customDimension);
}]);
```

Content Tracking

Content Tracking tracks how many times specific elements were rendered/visible. It can be used to measure if the ad placement was visible or if the user saw the end of an article.

To track content, it has to have the `data-track-content` attribute or `piwikTrackContent` CSS class attached to it.

Tracking all content impressions within a page

To track all content impression, you can use the `trackAllContentImpressions` function. If this function is invoked multiple times, it will not send duplicated data unless the `trackPageView` was used between invocations:

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

Tracking all visible content impressions

To track all visible content impressions you can use the `trackVisibleContentImpressions` function.

Code:

```
_paq.push(["trackVisibleContentImpressions", checkOnScroll, watchInterval]);
```

`checkOnScroll`
: **Optional** `boolean` If `true`, it will check new visible content impressions on the scroll event. Default: `true`.

> **Note:** It will not detect content blocks placed in a scrollable element.

`watchInterval`
: **Optional** `number` Interval, in milliseconds between checking for new visible content. Periodic checks can be disabled for performance reasons by setting `0`. Default value: `750`.

> **Warning:** Both options cannot be changed after the initial setup.

Example of usage:

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

Example of usage:

```
_paq.push(["trackVisibleContentImpressions", false, 500]);
```

Tracking only content impressions for specific page part

To track impressions on part of a webpage that will be populated after a page load, you can use the `trackContentImpressionsWithinNode`:

```
_paq.push(["trackContentImpressionsWithinNode", domNode]);
```

`domNode`
: **Required** `domNode` DOM element that will have impression DOM elements with `data-track-content` attribute

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

Example of usage:

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

Track interactions manually with auto detection

If you wish to trigger an interaction manually (for example on click event), you can do it using `trackContentInteractionNode`, just add this code in the action you want to track:

```
_paq.push(["trackContentInteractionNode", domNode, contentInteraction]);
```

domNode
: **Required** `domNode` Node marked as content block or containing content blocks. If no content block is found - nothing will be tracked.

contentInteraction
: **Optional** `string` Name of interaction (e.g. `"click"`). Default value: `"Unknown"`.

Example of use

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

Track impression manually

If you wish to trigger tracking impressions entirely manually, you can use the `trackContentImpression`:

```
_paq.push(["trackContentImpression", contentName, contentPiece, contentTarget]);
```

contentName
: **Required** `string` Name of Content Impression

contentPiece
: **Required** `string` Name of Content Impression Piece

contentTarget
: **Required** `string` URL of Content Impression Target

Example of use:

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

Track user interaction manually

If you wish to trigger tracking interactions entirely manually, you can use the `trackContentInteraction`. Use it as a function inside listener on event:

```
_paq.push(["trackContentInteraction", contentInteraction, contentName, contentPiece,␣
↪contentTarget]);
```

contentInteraction
: **Required** `string` Name of interaction (e.g. `"click"`).

contentName
: **Required** `string` Name of Content Impression

contentPiece
: **Required** `string` Name of Content Impression Piece

contentTarget
Required `string` URL of Content Impression Target

Example of use:

```
_paq.push(["trackContentImpression", "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 linking `contentName`. It does not map automatically as `trackContentInteractionNode`.

Download and Outlink Tracking

Tracking Outlink

To enable the Download & Outlink tracking, run:

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

just after the first `trackPageView` or `trackEvent`

Note: All Outlinks are tracked automatically. As `enableLinkTracking` is part of the default snippet.

Ignoring alias domains

To ignore internal outlinks from alias domains, use the `setDomains` function to define internal domains and subdomains, you can use the wildcard:

```
_paq(["setDomains", domains]);
```

domains
Required `array` Domains written as strings, * are accepted.

Example of usage:

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

Force Tracking using CSS class

To track clicking a link as an outlink using the CSS class, simply add the `piwik_link` class to the link element. It will then be considered as an outlink, even if it points to the same domain.

This class name can be changed, use `setLinkClasses` to define which CSS class should be tracked:

```
_paq.push(["setLinkClasses", className]);
```

className
Required `string` CSS class that should be tracked instead of `piwik_link`

Example of usage:

```
_paq(["setLinkClasses", "track-this-link"]);
```

Force Tracking using JS function

If you wish to use JS to force the outlink to be tracked, you can add the `trackLink` function on element `onClick` attribute:

```
_paq.push(["trackLink", linkAddress, "link", dimensions]);
```

`linkAddress`
Required `string` Address that link points to.

`dimensions`
Optional `object` *Custom dimension* that should be tracked with this action. Can be multiple dimensions. Written as object property using `dimension{ID}` notation.

Example:

```
{
   dimension1: "example value",
   dimension2: "example value"
}
```

Example of usage

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

Tracking Downloads

Default extensions recognized as download

The following extensions are tracked as download by default:

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

Adding extension to default extensions

You can add an extension to the default extensions list using the `addDownloadExtensions` function:

```
_paq.push(["addDownloadExtensions", extensions]);
```

`extensions`
Required `string|Array<string>` Extensions separated by `|` for example `"7z|apk|mp4"` can also be written as an Array, for example: `["7z","apk","mp4"]`

Example of usage:

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

Replacing default extensions list

Default extensions list can be overwritten using the `setDownloadExtensions` function:

```
_paq.push(["setDownloadExtensions", extensions]);
```

`extensions`
Required `string|Array<string>` Extensions separated by `|` for example `"7z|apk|mp4"` can also be written as an Array, for example: `["7z","apk","mp4"]`

Example of usage:

```
_paq.push(["setDownloadExtensions", "7z|apk|mp4"]);
```

Force Tracking download using CSS class

To track clicking a link as a download using css class simply add the `piwik_download` class to link element.

This class name can be changed, use `setDownloadClasses` to define which CSS class should be tracked:

```
_paq.push(["setDownloadClasses", className]);
```

`className`
Required `string` CSS class that should be tracked instead of `piwik_download`

Example of usage:

```
_paq(["setDownloadClasses", "track-this-link-for-download"]);
```

Force Tracking download using JS function

If you wish to use JS to force tracking download, you can add `trackLink` function on element `onClick` attribute:

```
_paq.push(["trackLink", linkAddress, "download", dimensions]);
```

`linkAddress`
Required `string` Address that link points to.

`dimensions`
Optional `object` *Custom dimension* that should be tracked with this action. Can be multiple dimensions. Written as object property using `dimension{ID}` notation.

Example:

```
{
   dimension1: "example value",
   dimension2: "example value"
}
```

Example of usage

```
<button onClick="function(){_paq.push(['trackLink', 'http://www.example.com/example.
↪xrt', 'download']);}">
    Click me!
</button>
```

Setting link tracking delay

After each outbound/download link click, there is a small delay introduced, after which the browser navigates to the new URL. This ensures there is enough time to track link interactions. That delay is set by default to 500ms. To modify it you can use the `setLinkTrackingTimer` function:

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

time
: **Required** `number` Time in ms between user action (click) and changing a website (for outlink) or downloading a file.

Disabling tracking

You can disable download and outlink tracking for links using CSS classes, simply add `piwik_ignore` css class.

To disable using CSS class you can use `setIgnoreClassess` function:

```
_paq.push(["setIgnoreClasses", className);
```

className
: **Required** `string|Array<string>` Css class name that will be ignored, can be written as Array with CSS classes.

User ID Management

User ID enables merging user data that is collected between many devices and browsers.

You must provide unique user-id for every user. To set user ID for tracked data use `setUserId` function:

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

userID
: **Required** `string` Unique, non-empty permanent ID of the user in application.

Miscellaneous

Custom page name

We are using the current page URL as the page title. To change this use the `setDocumentTitle` function:

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

title
: **Required** `string` Title to show instead of URL

Example of usage:

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

Measuring user time spent on web page

When the user will enter a single page during a visit, we will assume that his total time spent on the website was 0 ms. To measure that time more accurately you can use the `enableHeartBeatTimer` function:

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

beat
: **Required** `number` Time in seconds between cyclical heartbeat requests, default `15`

Example of usage:

```
_paq.push(["enableHeartBeatTimer", 50]);
```

Tracking internal searches

To track search requests on your site use the `trackSiteSearch` function:

```
_paq.push(["trackSiteSearch", keyword, category, searchCount, dimensions]);
```

keyword
: **Optional** `string` Keyword that was searched

category
: **Optional** `string` Category seleted in search engine - you can set it to false when not used.

searchCount
: **Optional** `number` Results on the results page - you can set it to false when not used.

dimensions
: **Optional** `object` *Custom dimension* that should be tracked with this action. Can be multiple dimensions. Written as object property using `dimension{ID}` notation.

  Example:

  ```
  {
     dimension1: "example value",
     dimension2: "example value"
  }
  ```

Example of usage:

```
_paq.push(["trackSiteSearch", "test", false, 20]);
```

2.2 Tracker Object Functions

This document describes all the functions available for the Tracker object and how to create its instances. This enables users to track multiple Trackers at once.

2.2.1 Accessing Tracker Object

To access Tracker object instance you must use the `Piwik.getTracker` function

`Piwik.`**`getTracker`**(*trackerUrl*, *siteId*)
Getter for Analytics Tracker instance.

Arguments

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

Returns Analytics Tracker instance

To access internal instance of the Tracker used for asynchronous tracking you must use the `Piwik.getAsyncTracker` function

`Piwik.`**`getAsyncTracker`**(*trackerUrl*, *siteId*)
Getter for Analytics Tracker instance.

Arguments

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

Returns Analytics Tracker instance

2.2.2 Tracking functions

`trackPageView`([*customPageTitle*])
Tracks a visit on the page that the function was run on.

Arguments

- **`customPageTitle`** (*`string`*) – **Optional** Custom page title, for example `document.title`

`trackEvent`(*category*, *action*[, *name*, *value*])
Tracks events that should not trigger on page loading, but only when user performs an action

Arguments

- **`category`** (*`string`*) – **Required** Category of event.
- **`action`** (*`string`*) – **Required** Event action, for example `"link click"`.
- **`name`** (*`string`*) – **Optional** Event name, for example `"Cancel button"`.
- **`value`** (*`string`*) – **Optional** Event value.

`trackGoal`(*idGoal*[, *customRevenue*, *customData*])
Manually tracks goal (conversion).

Arguments

- **`idGoal`** (*`int/string`*) – **Required** Id of goal.
- **`customRevenue`** (*`int/float`*) – **Optional** Revenue value
- **`customData`** (*`mixed`*) – **Optional** Object with *Custom dimensions*.

`trackSiteSearch`(*keyword*[, *category*, *resultCount*])
The function that tracks internal site searches.

Arguments

- **keyword** (*string*) – **Required** String containing keyword that was searched.
- **category** (*string/boolean*) – **Optional** String with category selected in search engine, can set it to false when not used.
- **searchCount** (*number/boolean*) – **Optional** Number of results on the results page, can be set to false when not used.

enableHeartBeatTimer (*delay*)
When the user will enter a single page during a visit, we will assume that his total time spent on the website was 0 ms. To measure that time more accurately you can use the `enableHeartBeatTimer` function

Arguments

- **delay** (*number*) – **Required** Time in seconds between cyclical heartbeat requests, default `15`

enableCrossDomainLinking ()
The function that will enable cross domain linking. That way visitors across domains will be linked.

setCrossDomainLinkingTimeout (*timeout*)
The function will change default time in which two visits across domains will be linked.

Arguments

- **timeout** (*number*) – **Required** Time in seconds in which two visits across domains will be linked. Default is `180`.

2.2.3 Ecommerce tracking

addEcommerceItem (*productSKU*[, *productName*, *productCategory*, *price*, *quantity*])
The function that adds ecommerce item, can be used when adding and removing items from cart.

Arguments

- **productSKU** (*string*) – **Required** String with product stock-keeping unit.
- **productName** (*string*) – **Optional** String with product name.
- **productCategory** (*Array<string>|string*) – **Optional** Product category, can be written as Array with up to 5 elements.
- **price** (*string*) – **Optional** String with product price.
- **quantity** (*string*) – **Optional** String with product quantity.

trackEcommerceOrder (*orderId*, *orderGrandTotal*[, *orderSubTotal*, *orderTax*, *orderShipping*, *orderDiscount*])
The function that tracks Ecommerce order, also tracks all items previously added.

Arguments

- **orderId** (*string*) – **Required** Unique order ID.
- **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 costs.

- **orderDiscount** (*number*) – **Optional** Order discount amount.

trackEcommerceCartUpdate(*grandTotal*)
The function that tracks the shopping cart value. Use this each time there is a change in cart as the last function after adding cart items.

Arguments

- **grandTotal** (*number*) – **Required** Order Revenue grand total - tax, shipping and discount included.

setEcommerceView(*productSKU*[, *productName*, *productCategory*, *productPrice*])
The function to track product or category page view, must be followed by the `trackPageView` function.

Arguments

- **productSKU** (*string*) – **Required** String with product stock-keeping unit.
- **productName** (*string*) – **Optional** String with product name.
- **productCategory** (*Array<string>|string*) – **Optional** Product category, can be written as Array with up to 5 elements.
- **price** (*string*) – **Optional** String with product price.

2.2.4 Custom variables

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

setCustomVariable(*index*, *name*, *value*, *scope*)
The function that sets a custom variable to be used later.

Arguments

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

deleteCustomVariable(*index*, *scope*)
The function that will delete a custom variable.

Arguments

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

getCustomVariable(*index*, *scope*)
The function that will return the value of custom variable.

Arguments

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

storeCustomVariablesInCookie()
The function will enable storing `"visit"` type custom variables in a first party cookie. That will enable getting them via the `getCustomVariable` function.

2.2.5 Custom dimensions

setCustomDimension(*customDimensionId*, *customDimensionValue*)
The function that sets a custom dimension to be used later.

Arguments

- **customDimensionId** (*string*) – **Required** Id of custom dimension.
- **customDimensionValue** (*string*) – **Required** Value of custom dimension.

deleteCustomDimension(*customDimensionId*)
The function that will delete a custom dimension.

Arguments

- **customDimensionId** (*string*) – **Required** Id of custom dimension.

getCustomDimension(*customDimensionId*)
The function that will return the value of custom dimension.

Arguments

- **customDimensionId** (*string*) – **Required** Id of custom dimension.

2.2.6 Content Tracking

Impressions

trackAllContentImpressions()
The function that will scan DOM for content blocks and tracks impressions after all page will load.

trackVisibleContentImpressions([*checkOnScroll*, *watchInterval*])
The function that will scan DOM for all visible content blocks and will track these impressions.

Arguments

- **checkOnScroll** (*boolean*) – **Optional** Enables tracking content blocks that will be visible after scroll event.
- **watchInterval** (*number*) – **Optional** Interval, in milliseconds between checking for new visible content. Periodic checks can be disabled for performance reasons by setting `0`. Default value: `750`.

trackContentImpressionsWithinNode(*domNode*)
The function that will scan domNode (with its children) for all content blocks and will track impressions.

Arguments

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

trackContentImpression(*contentName*, *contentPiece*, *contentTarget*)
The function that manually tracks content impression.

Arguments

- **contentName** (*string*) – **Required** String containing name of Content Impression.
- **contentPiece** (*string*) – **Required** String containing name of Content Impression Piece.

- **contentTarget** (*string*) – **Required** String containing URL of Content Impression Target.

logAllContentBlocksOnPage()
The function that will print all content blocks in the console for debugging purposes.

Interactions

trackContentInteractionNode(*domNode*[, *contentInteraction*])
The function that tracks interaction within domNode. This can be used as a function inside the onClick attribute.

Arguments

- **domNode** (*domNode*) – **Required** Node marked as content block or containing content blocks. If no content block will be found - nothing will be tracked.
- **contentInteraction** (*string*) – **Optional** Name of interaction (e.g. `"click"`). Default value: `"Unknown"`.

trackContentInteraction(*contentInteraction*, *contentName*, *contentPiece*, *contentTarget*)
The function that tracks content interaction using the given data.

Arguments

- **contentInteraction** (*string*) – **Required** Name of interaction (e.g. `"click"`).
- **contentName** (*string*) – **Required** Name of Content Impression.
- **contentPiece** (*string*) – **Required** Name of Content Impression Piece.
- **contentTarget** (*string*) – **Required** URL of Content Impression Target.

2.2.7 Download and Outlink Tracking

trackLink(*url*, *linkType*[, *customData*, *callback*])
The function that will manually track downloads or outlinks, depending on type.

Arguments

- **url** (*string*) – **Required** Address that link points to.
- **linkType** (*string*) – **Required** Type of link, if is set to `"link"` it will track an outlink, if it is set to `"download"` it will track a download.
- **customData** (*object*) – **Optional** Object containing *Custom dimension* that should be linked to tracked link.
- **callback** (*function*) – **Optional** The function that should be triggered after tracking link.

Tracking Outlink

enableLinkTracking(*enable*)
The function that will register all links as trackable (left and middle mouse buttons are being treated the same, right mouse button is treated as "open in a new tab").

Arguments

- **enable** (*boolean*) – **Required** Set it to true to track links, false to disable tracking.

setLinkClasses (*classes*)
The function that sets classes to be treated as outlinks (`piwik-link` is the default one).

Arguments

- **classes** (*array/string*) – **Required** String containing CSS class, can be written as array of strings.

Tracking Downloads

Default extensions recognized as download

The following extensions are tracked as download by default:

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

setDownloadClasses (*classes*)
The function that sets classes to be treated as downloads (`piwik_download` is the default one).

Arguments

- **classes** (*array/string*) – **Required** String containing CSS class, can be written as array of strings.

setDownloadExtensions (*extensions*)
The function that will set a list of file extensions that will automatically be recognized as a download action.

Arguments

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

addDownloadExtensions (*extensions*)
The function that will add extensions to a list of known extensions to be automatically recognized as a download action.

Arguments

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

removeDownloadExtensions (*extensions*)
The function that will remove extensions from a list of known extensions to be automatically recognized as a download action.

Arguments

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

setLinkTrackingTimer (*time*)
The function that will set delay between tracking and download;

Arguments

- **time** (*number*) – **Required** Delay between tracking and download, written in miliseconds.

getLinkTrackingTimer()
The function that will return delay between tracking and download.

Disabling tracking

setIgnoreClasses(*classes*)
The function that will set classes to be ignored in tracking download and outlinks.

Arguments

- **classes** (*array/string*) – **Required** String containing CSS class, can be written as array of strings.

2.2.8 User ID and Visitor ID Management

User ID

getUserId()
The function that will return user ID.

setUserId(*userId*)
The function that will set user ID to this user.

Arguments

- **userId** (*string*) – **Required** Unique, non-empty string preserved for each user.

Visitor ID

getVisitorId()
The function that will return 16 characters ID for the visitor.

getVisitorInfo()
The function that will return visitor information in an array:

- new visitor flag indicating new (`1`) or returning (`0`) visitor
- visitor ID (UUID)
- first visit timestamp (Unix epoch time)
- previous visit count (`0` for first visit)
- current visit timestamp (Unix epoch time)
- last visit timestamp (Unix epoch time or `''` if N/A)
- last e-commerce order timestamp (Unix epoch time or `''` if N/A)

2.2.9 Tracking cookies management

Cookies that are used by analytics are first party cookies.

disableCookies()
The function that will disable all first party cookies. Existing ones will be deleted in the next page view.

deleteCookies()
The function that will delete existing tracking cookies after the next page view.

hasCookies()
The function that will return `true` if cookies are enabled in this browser.

setCookieNamePrefix(*prefix*)
The function that will set the prefix for analytics tracking cookies. Default is `"_pk_"`

Arguments

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

setCookieDomain(*domain*)
The function that will set 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*)
The function that will set the analytics tracking cookies path.

Arguments

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

setSecureCookie(*bool*)
The function that will toggle the Secure cookie flag on all first party cookies (if you are using HTTPS).

Arguments

- **bool** (*boolean*) – **Required** If set to true it will add Secure cookie flag.

setVisitorCookieTimeout(*seconds*)
The function that will set the expiration time of visitor cookies.

Arguments

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

setReferralCookieTimeout(*seconds*)
The function that will set the expiration time of referral cookies.

Arguments

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

setSessionCookieTimeout(*seconds*)
The function that will set the expiration time of session cookies.

Arguments

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

2.2.10 Tracker Configuration

setDocumentTitle([*title*])
The function that will set the document tile that is being sent with tracking data.

Arguments

- **title** (*string*) – **Optional** String that will override default `document.title`

setDomains(*domains*)
The function that will set an array of domains to be treated as local. Sub-domain wildcards are supported (e.g. `*.example.com`).

Arguments

- **domains** (*array<string>*) – **Required** Array of hostnames written as strings.

setCustomUrl(*customUrl*)
The function that will override tracked page URL. Tracker will use current page URL if custom URL was not set.

Arguments

- **customUrl** (*string*) – **Required** Value that will override default URL.

setReferrerUrl(*url*)
The function that will override the detected HTTP referrer.

Arguments

- **url** (*string*) – **Required** Value that will override HTTP referrer.

setApiUrl(*url*)
The function that will set the Analytic's HTTP API URL endpoint. Usually the root directory of analytics.

Arguments

- **url** (*string*) – **Required** Path to Analytic's HTTP API URL

getPiwikUrl()
The function that will return the Analytic's server URL.

getCurrentUrl()
The function that will return the current URL of the page. The custom URL will be returned if set.

discardHashTag(*enableFilter*)
The function that will set tracker to include or remove URL fragment identifier from tracked URLs.

Arguments

- **enableFilter** (*boolean*) – **Required** If set to true, URL fragment identifier will be removed from tracked URLs.

setGenerationTimeMs(*generationTime*)
The function that overrides DOM Timing API provided time needed to download the page.

Arguments

- **generationTime** (*number*) – **Required** Time that will take to download page, in milliseconds.

appendToTrackingUrl(*appendToUrl*)
The function that will append a custom string to the tracking URL.

Arguments

- **appendToUrl** (*string*) – **Required** String that will be added to the tracking URL.

setDoNotTrack(*enable*)
The function that will disable tracking users who set the Do Not Track setting.

Arguments

- **enable** (*boolean*) – **Required** When set to true tracking wont occur.

killFrame()
The function that will block a site from being iframed.

redirectFile(*url*)
The function that will force the browser to load URL if the tracked web page was saved as a file.

Arguments

- **url** (*string*) – **Required** URL that should be loaded.

getAttributionInfo()
The function that will return visitor attribution array (Referrer and Campaign data).

getAttributionCampaignName()
The function that will return the Attribution Campaign name.

getAttributionCampaignKeyword()
The function that will return the Attribution Campaign keywords.

getAttributionReferrerTimestamp()
The function that will return the Attribution Referrer timestamp.

getAttributionReferrerUrl()
The function that will return the Attribution Referrer URL.

setCampaignNameKey(*name*)
The function that will set campaign name parameters.

Arguments

- **name** (*string*) – **Required** Campaign name.

setCampaignKeywordKey(*keyword*)
The function that will set campaign keyword parameters.

Arguments

- **keyword** (*array<string>*) – **Required** Keyword parameters.

setConversionAttributionFirstReferrer(*bool*)
The function that will set if an attribute will convert to the first referrer

Arguments

- **bool** (*boolean*) – **Required** If set to true attribute will convert to the first referrer otherwise it will be converted to most recent referrer.

2.2.11 Advanced Usage

addListener(*domElement*)
The function will add a click listener to link element.

Arguments

- **domElement** (*DOMElement*) – **Required** Element that click will trigger logging the click automatically.

setRequestMethod(*method*)
The function that will set the request method.

Arguments

- **method** (*string*) – **Required** Method that will be used in requests. `"GET"` or `"POST"` are supported. The default is `"GET"`

setCustomRequestProcessing (*function*)
The function that will process the request content. The function will be called once the request (query parameters string) has been prepared, and before the request content is sent.

setRequestContentType (*contentType*)
The function that will set tracking requests `Content-Type` header. Used when tracking uses the `"POST"` method (set by `setRequestMethod`).

Arguments

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

2.3 HTTP API

Tracking HTTP API allows sending to analytics information about Visitors page views, events and visits.

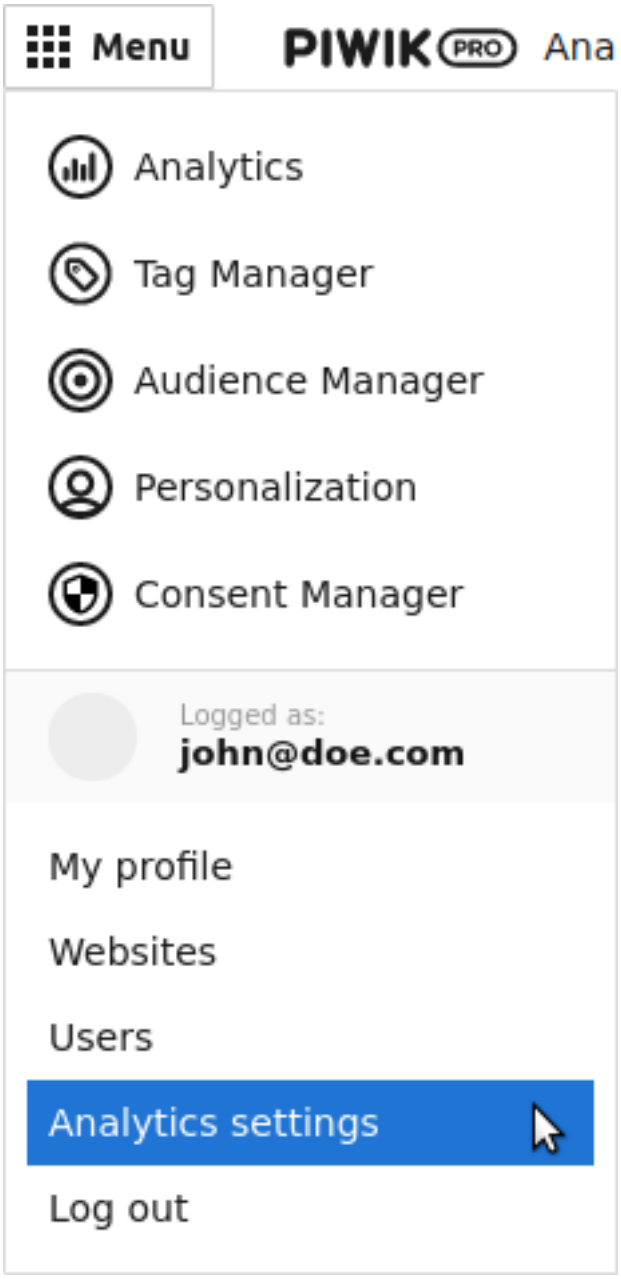
Deprecated since version 5.5.1: Endpoint `/piwik.php` is moved to `/ppms.php`. The old endpoint still works, but its support will be disabled at some point.

2.4 Reporting API

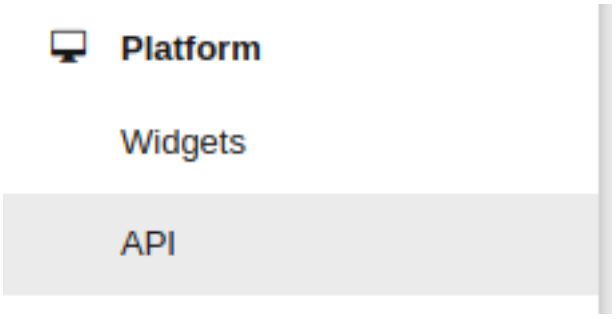
This API gives access to your analytics reports.

A short description of all available reports is available in *Analytics*. Follow these instructions to access it:

1. Login into your PPAS instance
2. Go to *Menu* > *Analytics settings*.



3. Select in the left menu *Platform* > *API*.



CHAPTER 3

Audience Manager

3.1 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*.

3.1.1 Loading snippet

Add the following snippet on your page to start using this API. It should be added just before the first API usage.

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]||{};
↪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.

3.1.2 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

3.1.3 Commands

All commands work in context of the current *user*. 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 *user* 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 *user* 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 *user* 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 *user* 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 *user* profile *attributes*. The *user* 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 *user 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>>*) – **Required** Object containing *user attributes* divided by source.
  - *analytics* - `Object<string,string>` Contains *analytics attributes* about the *user* (e.g. browser name, browser version, country).
  - *attributes* - `Object<string,string>` Contains *custom attributes* about the *user* (e.g. first name, last name, email).

  Example:

```
{
    "analytics": {
        "browser_name": "chrome",
        "country": "us"
    },
    "attributes": {
        "first_name": "James",
        "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 *user 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>` Object containing *attributes* to update:

- key - *attribute* name
- value - *attribute* value

Example:

```
{
    "favourite_color": "black",
    "drink": "Martini"
}
```

options
Optional `object` Object that can specify additional *user identifiers* and callback functions.

Example:

```
{
    "user_id": user_id,
    "device_id": device_id,
    "email": email,
    "onFulfilled": onFulfilled,
    "onRejected": onRejected
}
```

user_id
Optional `string` If the *application* lets *user* 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 *user* 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"
  ```

3.2 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 field in the form. The *attribute* name is generated from input tag (HTML tag's `name` attribute). Inputs without a name are ignored.

3.2.1 Supported browsers

All modern browsers: Chrome, Firefox, Safari, Edge. Internet Explorer from version 8 and above.

3.2.2 Private information

Form Tracker is trying to automatically detect fields containing *user's* private information and ignores them. 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).
- Data from ignored fields (see *Optional configuration*).

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 ignore such forms using the *Optional configuration*.

3.2.3 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).

```
<script>
    (function(a,d,g,h,b,c,e){a[b]=a[b]||{};a[b][c]=a[b][c]||{};
↪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)})
```

(continues on next page)

(continued from previous page)

```
    (window,document,"script","https://your-instance-name.piwik.pro/audiences/static/
↪widget/audience-manager.form.min.js","ppms","am","form");

    ppms.am.form("set", "WebsiteID", "XXX-XXX-XXX-XXX-XXX");
    ppms.am.form("set", "TrackerUrl", "https://your-instance-name.piwik.pro/audiences/
↪tracker/");
    ppms.am.form("set", "StaticUrl", "https://your-instance-name.piwik.pro/audiences/
↪static/widget/");
</script>
```

Note: Usually it is recommended to use the **HTTPS** protocol in the URLs mentioned here, but if support for **legacy IE browsers** (8 and 9) is required and some pages containing forms are served via **HTTP** protocol - it is necessary to use the same protocol in snippet URLs as the source page. The easiest way to do that would be to remove the protocol from `TrackerUrl` and `StaticUrl` (e.g. `//your-instance-name.piwik.pro/audiences/tracker/`).

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 basic configuration of Form Tracker `ppms.am.form`.

When the loading snippet is added on the page without any *Optional configuration*, the Form Tracker will gather information from all forms submitted on the page. It is possible to modify this behavior by configuring optional rules at the end of the snippet. You can do that by using the command queue (`ppms.am.form`) immediately after step 3 (see *Optional configuration*).

3.2.4 Command queue

The loading snippet creates the following global function:

`ppms.am.`**`form`**(*command*, *...args*)

Audience Manager Form Tracker 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

3.2.5 Optional configuration

These commands allow you to limit the scope of forms watched by the Form Tracker.

Ignore form

You can force the Form Tracker to ignore the selected form as a whole or specific fields in it. The Form Tracker will not gather any data from fields of a form specified in this way. You can ignore multiple forms by configuring the ignore rule multiple times (separately for each form).

Code:

```
ppms.am.form("ignore", form_id, field_names);
```

`form_id`
Required `string` `id` attribute of ignored `<form>` tag.

Example:

```
"payment-form"
```

`field_names`
Optional `Array<string>` List of `name` attributes of ignored `<input>` or `<textarea>` tags in the ignored form. If this parameter is not provided, all fields in the form will be ignored.

Example:

```
["street", "post-code", "city"]
```

Note: If this parameter is empty array (`[]`) no field will be ignored.

Note: This configuration may be called multiple times and its effects will be cumulative:

- If calls specify different `form_id` - each form will be ignored accordingly.
- If multiple calls specify same `form_id`:
  - If any of the calls omit the `field_names` parameter - the whole form will be ignored.
  - If all calls specify the `field_names` - all fields specified across all calls will be ignored.

3.3 Public HTTP API

3.4 Authorized HTTP API

CHAPTER 4

Consent Manager

4.1 JavaScript API

4.1.1 Introduction

Consent Manager provide JavaScript API that allows the user to:

- get consent types
- get new consent types
- get consent settings
- set consent settings
- send data subject request

JavaScript API is implemented by providing global JavaScript objects queue responsible for executing command:

`ppms.cm.`**`api`**(*command*, *...args*)

`dataLayer.`**`push`**(*{event: command*, *...args}*)

dataLayer.push interface is only for backward compatibility and you can read more about this particular case below. We recommend `` `ppms.cm.api` ``.

param string command Command name.

param 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.

rtype undefined

4.1.2 Installation

Consent Manager is fully integrated with Tag Manager. If you have already installed asynchronous snippet and you are using API only from Tag Manager tags, you are able use JavaScript API without aby pitfalls.

Only one thing should be considered before using API is where you call commands. If your goal is to perform API method outside Tag Manager tags like in below example:

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="UTF-8">
    <title></title>
</head>

<!-- Start Piwik PRO Tag Manager code -->
<script>
    // Tag Manager async code snippet
</script>
<!-- End Piwik PRO Tag Manager code -->

<script>
    // API call outside Tag Manager injected manually
    ppms.cm.api(command, ...args);
</script>

<body>
    Rest content of document
</body>

</html>
```

When you need execute API in such manner, you should take care about Tag Manager snippet version. Because *ppms.cm.api* global object is initialized in snippet and/or in Tag Manager container, if you are using old version of Tag Manager snippet (PPAS version > 6.2), *ppms.cm.api* might be undefined until container will be loaded. So if you want use own scripts outside Tag Manager, you need update snippet to use *ppms.cm.api*, or use *dataLayer* interface if replacing snippet is not possible:

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="UTF-8">
    <title></title>
</head>

<!-- Start Piwik PRO Tag Manager code -->
<script>
    // Tag Manager async code snippet
</script>
<!-- End Piwik PRO Tag Manager code -->

<script>
    // API call outside Tag Manager injected manually
    dataLayer.push({event: command, ...args});
</script>

<body>
    Rest content of document
```

(continues on next page)

(continued from previous page)

```
</body>

</html>
```

4.1.3 Commands

All commands work in 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 always required. *onFailure(exception)* callback is optional and if is specified, any error object occurred will be passed as a argument. If not specified, error is reported directly on console output.

Get consent types

Fetches a list of consent types.

Code:

```
ppms.cm.api('getComplianceTypes', onFulfilled, onRejected);
dataLayer.push({'event': 'ppms.cm:getComplianceTypes', parameters: [onFulfilled,␣
↪onRejected]});
```

dataLayer.push interface is only for backward compatibility and you can read more about this particular case below. We recommend `ppms.cm.api`.

onFulfilled(*types*)
required The fulfilment 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, exception will be thrown in main stacktrace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception.

Get new consent types

Fetches a list of new consent types which were appearing after given consents.

Code:

```
ppms.cm.api('getNewComplianceTypes', onFulfilled, onRejected);
dataLayer.push({'event': 'ppms.cm:getNewComplianceTypes', parameters: [onFulfilled,␣
↪onRejected]});
```

dataLayer.push interface is only for backward compatibility and you can read more about this particular case below. We recommend `ppms.cm.api`.

onFulfilled(*types*)
required The fulfilment 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 compliance settings

Set compliance settings base on user decision. This API command might be useful when user interact with custom, extended UI that reacts on user approve/reject action. After successful, Consent Manager internally send consent settings to tracking server and force page view on tags.

Code:

```
ppms.cm.api('setComplianceSettings', settings, onFulfilled, onRejected);
dataLayer.push({'event': 'ppms.cm:setComplianceSettings', parameters: [settings,␣
↪onFulfilled, onRejected]});
```

dataLayer.push interface is only for backward compatibility and you can read more about this particular case below. We recommend `ppms.cm.api`.

settings
required The consent settings object.

Example:

```
{consents: {analytics: {status: -1}}}
```

Where *consent.analytics* is consent type and status indicate:

- *-1* - user does not interact, e.g. close consent popup without any decision
- *0* - user has rejected the consent
- *1* - user has approved the consent

onFulfilled()
required The fulfilment handler callback. This function is **required**.

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, exception will be thrown in main stacktrace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception.

Get compliance settings

Return current privacy settings. Might be useful for initializing custom decision view. When there is no decisions, just returns empty object. This state can be used to detect first time user interaction with consent mechanism.

Code:

```
ppms.cm.api('getComplianceSettings', onFulfilled, onRejected);
dataLayer.push({'event': 'ppms.cm:getComplianceSettings', parameters: [onFulfilled,␣
↪onRejected]});
```

dataLayer.push interface is only for backward compatibility and you can read more about this particular case below. We recommend `ppms.cm.api`.

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 fulfilment handler callback (called with result).

onRejected(*error*)
The rejection handler callback (called with error code). If not specified, exception will be thrown in main stacktrace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception.

Send data request

Command send data subject request to Consent Manager collector.

Code:

```
ppms.cm.api('sendDataRequest', request, onFulfilled, onRejected);
dataLayer.push({'event': 'ppms.cm:sendDataRequest', parameters: [request, onFulfilled,
↪ onRejected]});
```

dataLayer.push interface is only for backward compatibility and you can read more about this particular case below. We recommend `ppms.cm.api`.

request
required The subject data request.

Example:

```
{content: '', email: '', type: 'change_data|view_data|delete_data'}
```

onFulfilled()
: **required** The fulfilment handler callback.

onRejected(*error*)
: The rejection handler callback (called with error code). If not specified, exception will be thrown in main stacktrace.

Arguments

- **error** (*string|object*) – **Required** Error code or exception.

4.1.4 Example usage

Based on above listed commands there are many possibilities to implement custom consent gathering. Below is listed a simple implementation using jQuery library.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:;base64,iVBORw0KGgo=">
  <title>Piwik Pro Tag Manager Custom Consent Implementation</title>

  <!-- Start Piwik PRO Tag Manager custom consent css code -->
  <link rel="stylesheet" href="https://rawgit.com/djanix/jquery.switcher/master/dist/
↪switcher.css"/>
  <style>
    * {
      font-family: BlinkMacSystemFont, -apple-system, Roboto, Oxygen-Sans, Ubuntu,␣
↪Cantarell, "Helvetica Neue", sans-serif;
    }

    .consent-container {
      background: white;
      display: none;
      bottom: 0;
      position: fixed;
      width: 100%;
      border-top: 1px solid #e0e0e0;
      z-index: 10000;
      color: rgba(0, 0, 0, 0.7);
      box-sizing: border-box;
    }

    .consent-content {
      display: inline-flex;
      width: 100%;
    }

    .consent-left {
      flex: 1 0 0;
      flex-direction: column;
      border-right: 1px solid #e0e0e0;
      font-size: 14px;
      display: flex;
      justify-content: center;
```

(continues on next page)

(continued from previous page)

```
    align-items: center;
    text-align: center;
    padding: 0 30px;
  }

  .consent-right {
    flex: 3 0 0;
    box-sizing: border-box;
    display: flex;
    opacity: 0.9;
    padding: 30px;
    background-color: #f5f5f5;
    line-height: 16px;
  }

  .consent-opt-in-button {
    border-color: rgba(197, 103, 57, 1);
    background-color: rgba(252, 131, 72, 1);
    color: #fff;
    min-width: 120px;
    font-weight: 600;
    font-size: 16px;
    line-height: 17px;
    text-align: center;
    border: 1px solid;
    border-radius: 2px;
    outline: 0;
    cursor: pointer;
    padding: 9px 16px 9px 16px;
  }

  .consent-link-more {
    color: #107EF1;
    font-size: 14px;
    line-height: 16px;
    margin-top: 7px;
    text-decoration: none;
    display: inline-block;
  }

  .consent-bottom {
    max-height: 0;
    transition: max-height 0.5s;
  }

  .consent-items {
    box-sizing: border-box;
    position: relative;
  }

  .consent-items-container {
    display: flex;
    flex-direction: column;

  }

  .consent-items-text {
```

(continues on next page)

(continued from previous page)

```
    margin-left: 10px;
  }

  .consent-item {
    display: flex;
    height: 50px;
  }

  .consent-item-left {
    width: 25%;
    border-right: 1px solid #e0e0e0;
    box-sizing: border-box;
    display: flex;
    justify-content: space-between;
  }

  .consent-item-right {
    width: 75%;
    display: flex;
    align-items: center;
  }

  .consent-item-right-text {
    font-size: 14px;
    margin: 0 30px;
  }

  .consent-items-description {
    padding: 20px 0;
    max-height: 54px;
    display: inline-flex;
    width: 100%;
    border-top: 1px solid #e0e0e0;
    border-bottom: 1px solid #e0e0e0;
  }

  .consent-items-footer {
    padding: 20px 0;
    max-height: 54px;
    width: 100%;
    border-top: 1px solid #e0e0e0;
  }

  label {
    width: 100%;
    padding: 0 30px;
    box-sizing: border-box;
    font-weight: 500;
    line-height: 55px;
    cursor: pointer;
    margin: 0;
  }

  .consent-switcher {
    margin: 10px 10px 10px 0;
  }
```

(continues on next page)

(continued from previous page)

```
    .consent-blue {
      background: #107EF1;
      border: 1px solid #107EF1;
    }
  </style>
  <!-- End Piwik PRO Tag Manager custom consent css code -->
</head>
<body>

<!-- PUT HERE CONTAINER JS CODE -->

<!-- Start Piwik PRO Tag Manager custom consent javascript code -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.0.0/jquery.min.js"></
↪script>
<script src="https://rawgit.com/djanix/jquery.switcher/master/dist/switcher.js"></
↪script>

<div style="display: none; visibility: hidden;" data-template="consentitem">
  <div class="consent-item">
    <div class="consent-item-left">
      <div>
        <label>${name}</label>
      </div>
      <div class="consent-switcher">
          <input class="consent-checkbox" type="checkbox" name="consentValues" value="
↪${key}" />
      </div>
    </div>
    <div class="consent-item-right">
      <div class="consent-item-right-text">
        ${description}
      </div>
    </div>
  </div>
</script>

<script>
  var availableConsents = [
    {
      key: 'analytics',
      name: 'Analytics',
      description: 'We will store data in an aggregated form about visitors and their␣
↪experiences on our website. We use this data to fix bugs and improve the experience␣
↪for all visitors.'
    },
    {
      key: 'ab_testing_and_personalization',
      name: 'AB Testing',
      description: 'We will create a cookie in your browser to ensure consistency of␣
↪our A/B tests. A/B tests are small changes displayed to different groups of␣
↪visitors. We use the data to create a better experience for all visitors. We will␣
↪also use this cookie to personalize content for you.'
    },
    {
      key: 'conversion_tracking',
      name: 'Conversion Tracking',
      description: 'We will store data about when you complete certain actions on our␣
↪website to understand better how you use it. We use this data to improve your␣
↪experience with our site.'
```

(continues on next page)

(continued from previous page)

```
    },
    {
      key: 'marketing_automation',
      name: 'Marketing Automation',
      description: 'We will store data to create marketing campaigns for certain␣
↪groups of visitors.'
    },
    {
      key: 'remarketing',
      name: 'Remarketing',
      description: 'We will store data to show you our advertisements (only ours) on␣
↪other websites relevant to your interests.'
    },
    {
      key: 'user_feedback',
      name: 'User Feedback',
      description: 'We will store data in an aggregated form to analyze the␣
↪performance of our website\'s user interface. We use this data to improve the site␣
↪for all visitors.'
    },
    {
      key: 'custom_consent',
      name: 'Custom consent',
      description: 'Adjust this copy to your needs.'
    },
  ];

  var customConsentSolution = {
    isDetailsOpen: false,
    containerSelector: '#consent-container',
    consentBottomSelector: '#consent-bottom',
    consentLinkMoreSelector: '#consent-link-more',
    consentItemFooterSelector: '#consent-items-footer',
    switcherSelector: '.consent-checkbox',
    optInButton: '.consent-orange',
    sendConsentButtonSelector: '.consent-blue',
    itemsSelector: '.consent-items-container',
    consentTemplate: $('div[data-template="consentitem"]').text().split(/\$\{(.+?)\}/
↪g),
    switcherElement: null,

    init: function() {
      $(this.consentLinkMoreSelector).click(this.showDetails.bind(this));
      $(this.sendConsentButtonSelector).click(this.sendConsents.bind(this, false));
      $(this.optInButton).click(this.sendConsents.bind(this, true));
      this.loadConsentList();
    },

    show: function() {
      $(self.containerSelector).slideDown(100);
    },

    hide: function() {
      $(self.containerSelector).slideUp(100);
    },

    loadConsentList: function() {
```

(continues on next page)

(continued from previous page)

```
      ppms.cm.api('getNewComplianceTypes', function(types) {
        self = customConsentSolution;

        if (types.length > 0) {
          self.show();
        }

        $(self.itemsSelector).append(
          availableConsents
            .filter(function(element) {
              return types.join(',').indexOf(element.key) !== -1;
            })
            .map(function (item) {
              return self.consentTemplate.map(self.replaceTemplate(item)).join('');
            })
        ).ready(function() {
          self.switcherElement = $(self.switcherSelector).switcher();
        });

      }, function(e) {});
    },

    sendConsents: function(all) {
        var queryElements = {
          consents: {},
        };

        $.each(this.switcherElement, function(index, value) {
          queryElements.consents[$(value).val()] = {
            status: all ? 1 : +$(value).prop('checked'),
          };
        });

        ppms.cm.api('setComplianceSettings', queryElements, function() {
          self = customConsentSolution;
          self.hide();
        }, function() {

        });
    },

    showDetails: function() {
      var detailsScrollHeight = $(this.consentBottomSelector).prop('scrollHeight');
      var baseScrollHeight = $(this.containerSelector).prop('scrollHeight');
      var consentItemFooterHeight = $(this.consentItemFooterSelector).prop(
↪'scrollHeight');

      $(this.consentBottomSelector).css({
        maxHeight: baseScrollHeight + detailsScrollHeight + consentItemFooterHeight +
↪"px",
        overflowY: 'auto',
        display: 'block',
      });
    },

    replaceTemplate: function(props) {
        return function(token, i) { return (i % 2) ? props[token] : token; };
```

(continues on next page)

(continued from previous page)

```
    },
  };

  $(document).ready(customConsentSolution.init.bind(customConsentSolution));
</script>
<!-- End Piwik PRO Tag Manager custom consent javascript code -->


<!-- Start Piwik PRO Tag Manager custom consent html code -->
<div class="consent-container" id="consent-container">
  <div class="consent-content">
    <div class="consent-left">
      <button class="consent-opt-in-button consent-orange">Opt-in and let's go!</
↪button>
    </div>
    <div class="consent-right">
      <div>
        <h1>[IMPORTANT] You're invited!...</h1>
        <a class="consent-link-more" id="consent-link-more" href="#">Show more</a>
      </div>
    </div>
  </div>
  <div class="consent-bottom" id="consent-bottom">
    <div class="consent-items">
        <div class="consent-items-description">
          <div class="consent-items-text">
            ...to tell us how you want us to handle your data.
            We'll only use your data for purposes you consent to.
            Change your mind whenever, we'll adapt to your consent preferences and␣
↪data requests.
          </div>
        </div>
        <div class="consent-items-container"></div>
        <div class="consent-items-footer" id="consent-items-footer">
          <div class="consent-items-text">
            <button class="consent-opt-in-button consent-blue">Save choices</button>
          </div>
        </div>
    </div>
  </div>
</div>
<!-- End Piwik PRO Tag Manager custom consent code -->

</body>
</html>
```

CHAPTER 5

Platform

5.1 Authorized API guide

This page describes how to access Piwik PRO API which uses client credentials OAuth grant type for obtaining user token. All data is sent and received as JSON and it uses JSON API specification.

5.1.1 Obtaining token

If you want to access API for the first time you need to generate your API credentials which then allows you to request for a token that is used for authentication during communication with authorized API.

Generate API Credentials

- Login to your account using your email and password.
- Go to your profile (`Menu` then `My profile`).
- On this page click on `API Credentials` tab. This page allows you to manage all your API credentials.
- Click `Generate new credentials` which will result in new popup. Fill in your custom `credentials name`. Name must contains at least 3 characters.
- Copy your newly generated `CLIENT ID` and `CLIENT SECRET` because they **won't be available for you after dismissing this window.**

Create access token

Having generated your API Credentials, now you are ready for creating access token that will be used in communication with API. To do so, send POST request

```
POST /auth/token
```

Curl example:

```
curl 'https://<domain>/auth/token' -H "Content-Type: application/json" --data '{
    "grant_type": "client_credentials",
    "client_id": "your_generated_client_id",
    "client_secret": "your_generated_client_secret"
}'
```

The example response is:

```
{"token_type":"Bearer","expires_in":86400,"access_token":"your_access_token"}
```

Piwik PRO API tokens use jwt format.

Now, you can use obtained `access_token` for communication with Piwik PRO API. Field `expires_in` stands for time (in seconds) for token TTL. As token is a Bearer type, it must be **included in all API calls** within header.

```
Authorization: Bearer your-token-here
```

5.1.2 Deleting API Credentials

Once you want to revoke the possibility of generating API token using given `CLIENT ID` and `CLIENT SECRET`, go to to `My profile` and click `Delete` button on selected API credentials.

5.2 Users API

5.3 Apps API

5.4 Access Control API

CHAPTER 6

Glossary

Application Website or application tracked by PPAS.

App ID PPAS *application* identificator (previously **website ID**, **site ID** or **idSite**).

User Visitor on tracked *application*.

Analytics ID ID assigned to *user* by *Analytics* for the duration of *Analytics* session. It is stored in browser cookie.

User ID Permanent ID assigned to *user* by *application* (e.g. username). You can read more about it here.

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 *user* ID (e.g. *analytics ID*, *user ID*, *device ID* or email).

Visit Period of continuous *user* activity on *application*. It ends in the following situations:

- after a period of inactivity (option set to 30 minutes by default)
- at midnight (option enabled by default)
- on campaign change (option enabled by default)
- when HTTP referrer points to different website (option disabled by default)

Audience Named set of *attribute* conditions used to define a group of *users* matching them.

Attribute Named value assigned to *user* profile.

Attribute whitelist List of *user attributes* that are publicly available via Audience Manager API.

> **Note:** It is still necessary to identify the *user* with his *analytics ID* to access this information.

PII Personally Identifiable Information.

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*.

Custom attribute *Attribute* generated from value provided by source other than *Analytics* (e.g. *Form Tracker*, *SDK*). 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*.

Analytics PPAS component gathering statistics about each *user* of the *application* (previously **Piwik**).

Index

Symbols

A

C

D

E

G

H

I

K

L

O

P